НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Спосіб забезпечення безпеки у мобільних мережах»

Виконав (-ла):
студент (-ка) IV курсу, групи ІО-61
Воробйов Анатолій Олександрович _____

Керівник:
Професор кафедри ОТ, д.т.н.,
Кулаков Юрій Олексійович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент кафедри СПіСКС, к.т.н.,
Орлова Марія Миколаївна _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Воробйова Анатолія Олександровича

1. Тема проекту «Спосіб забезпечення безпеки у мобільних мережах», керівник проекту Кулаков Юрій Олексійович, професор, д.т.н., затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проекту 2020р.

3. Вихідні дані до проекту: технічне завдання, теоретичні дані.

4. Зміст пояснювальної записки: опис предметної області, опис проблем, існуючих рішень, формування власного рішення, вибір та порівняння засобів для написання додатку, опис архітектури додатку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П. проф.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми диплому	01.09.2019	
2.	Вивчення та аналіз завдання	12.09.2019	
3.	Розробка архітектури та загальної структури системи	05.10.2019	
4.	Розробка структур окремих підсистеми	15.12.2019	
5.	Програмна реалізація	20.03.2020	
6.	Оформлення пояснювальної записки	01.04.2020	
7.	Предзахист	26.05.2020	
8.	Захист	25.06.2020	

Студент Анатолій ВОРОБЙОВ

Керівник Юрій КУЛАКОВ

Анотація

В даній дипломній роботі проведений аналіз засобів забезпечення безпеки у мобільних мережах. Виявлені головні недоліки існуючих реалізацій, та запропонований підхід у якого данні недоліки відсутні. Проведене моделювання атак на мережу для оцінки ефективності запропонованого способу. Реалізований запропонований спосіб для забезпечення безпеки у мобільних мережах.

Annotation

This Bachelor's work includes the analysis of means of safety in mobile networks. The main shortcomings of the existing implementations were identified and the proposed approach lacking these shortcomings. Modeled network attacks to evaluate the effectiveness of the proposed method. A proposed method for securing mobile networks has been implemented.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ДП 045430. 00.000 ВП	Відомість проекту	1	
3	А4	ДП 045430. 01.000 ТЗ	Технічне завдання	3	
4	А4	ДП 045430. 02.000 ПЗ	Пояснювальна записка	63	
5	А3	ДП 045430. 03.000 Д1	Принципова схема програмного забезпечення	1	
6	А3	ДП 045430. 04.000 Д2	Структурна схема мережі	1	
7	А3	ДП 045430. 05.000 Д3	Функціональна схема блокчейну з брандмауером	1	
8	А4	ДП 045430. 06.000 ДА	Лістинг програми	16	

					ДП 045430. 00.000 ВП	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Воробйов А.О.			Спосіб забезпечення безпеки у мобільних мережах Відомість проекту	Літ. / Арк: 1 / Аркушів: 1
Перевірив		Кулаков Ю.О.				
Реценз.						НТУУ КПІ, ФІОТ, ІО-61
Н. Контр.		Сімоненко В. П.				

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Спосіб забезпечення безпеки у мобільних мережах»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку курсу «Інженерія програмного забезпечення». Область застосування: застосування у мобільних мережах для забезпечення безпеки.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка нового методу по забезпеченню безпеки у мобільних мережах .

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, публікації в Інтернеті з даних питань.

5. **ТЕХНІЧНІ ВИМОГИ**

5.1. Вимоги до розробляємого продукту

- Система повинна визначати втручання в мережу
- Система повинна повідомляти про втручання в мережу адміністратора мережі.

5.2. Вимоги до програмного забезпечення

- Операційна система Linux 18 та новіші версії.
- Наявність на комп’ютері NodeJS не нижче версії 11.0.

5.3. Вимоги до апаратної частини

- Оперативної пам’яті не менше 8 ГБайт.
- Вільне місце на жорсткому диску не менше 200 Мбайт.

Пояснювальна записка
до дипломного проекту
на тему: «Спосіб забезпечення безпеки у мобільних мережах»

Київ – 2020 року

ЗМІСТ

ВСТУП

На сьогоднішній найбільш актуальною системою зв'язку є мобільні мережі. За останні роки найбільш вживаними є мережі третього, четвертого, п'ятого поколінь. Метою мереж третього покоління, або 3G було підвищення швидкості передачі даних, технологія основана на TDMA, GSM і CDMA. Через стрімкий ріст мобільного трафіку операторам мобільного зв'язку довелось впровадити LTE мережі, також відомі як четвертого покоління мереж. Головною метою четвертого покоління стало забезпечення безпеки, підвищення швидкості передачі даних. У четвертого покоління були фіксовані IP адрес користувачів, що і є одним з недоліків, так як, поява хмарних обчислень та мереж розповсюдження контенту (CDN), виявила недоліки масштабування, які обмежують еволюцію такої мережі. Тому на даний момент розробляється вже п'ятого покоління мереж. Нові мережі мають три напрями застосування. Першим напрямом є надання безперебійної роботи користувачів. Другий покриває централізоване постачання послуг для транспортних систем. Останнім напрямом є забезпечення масштабованості, енергоефективності.

Проблемами традиційних мереж є залежність від виробника обладнання, складність керування великими мережами, складність налагодження, складність впровадження нових ідей, не ефективне використання ресурсів мережі. З часом росте складність таких мереж, тому ці проблеми стають вагомішими. Маршрутизація у традиційних мережах - це розподілений процес, під час якого робоча топологія мережі визначає маршрути всіма пристроями. При відмові каналу в такій мережі, маршрутизатор сповістить під'єднанні до нього сусідні маршрутизатори о виниклій проблемі, і всі вони займуться незалежно розробкою нових маршрутів. Чим більший розмір мережі, тим довше буде проходити повне пере налаштування. Також цей процес не є детермінований, тому при повторній відмові немає гарантій що система перейде в те й самий стан. Тому високий ріст розвитку сервісів, заміна

клієнт-серверної архітектури на центри обробки даних (ЦОД) і хмари підвищили показники навантаження на мережу. Данні фактори впливають на швидкість впровадження нових ідей, збільшують витрати на підтримку. Тому розробляються нові підходи керування мережею головним питанням яких є забезпечення безпеки.

РОЗДІЛ 1
ОГЛЯД МЕТОДІВ МАРШУТИЗАЦІЇ В МОБІЛЬНИХ МЕРЕЖАХ

1.1 Аналіз систем для мобільних мереж

Через стрімке зростання кількості клієнтів мережі, навантаження на неї почало швидко рости[1]. Традицій мережі стали слабким елементом, так як переналаштування всієї мережі займає великий проміжок часу, впровадження нових ідей, та застосування стороннього обладнання висвітлюють основні проблеми традиційного підходу. Для того щоб вирішувати ці проблеми, почали розробляти підходи по підвищенню пропускної спроможності мережі, та зменшенню часу її реакції. За останні роки такими підходами стали SDN та NFV. Вони надають необхідні компоненти для покращення мережі.

1.1.1 SDN підхід

Для вирішення проблем традиційних мереж існує концепція програмно-конфігуруванні мережі (SDN) - це централізація керування мережею завдяки контролеру, визначення маршрутів для пакетів через фізичні прилади в рамках одного з’єднання. Відбувається перехід від роздільного керування мережевим обладнанням до управління мережею в цілому. Контролер проводить моніторинг завантаженості мережі і приймає рішення по розвантаженню вузлів та оперативним чином вирішує проблеми які можуть виникнути, наприклад при розриві каналів[1]. Головною перевагою такого підходу є те що відбувається оптимізація прокладання маршруту, так як формуванням маршруту займається не кожен пристрій окремо, а цим займається контролер. SDN-мережа представляє абстракцію (рис. 1.1), вся архітектура поділена на два сектори, це північний та південний інтерфейс і контролер між ними[1]. Реалізація цієї системи виглядає так: спеціалізоване програмне забезпечення яке працює на окремому сервері або на групі серверів в залежності від масштабу мережі, називається контролер, при цьому у мережних елементів відбирають функцію управління, вони виконують суто базові завдання. Така архітектура дозволяє виділити рівень управління і зробити його

програмованим. Основною ідеєю є збільшення ефективності мережі і надати можливість гнучкої адаптації до різних вимог прикладного характеру.

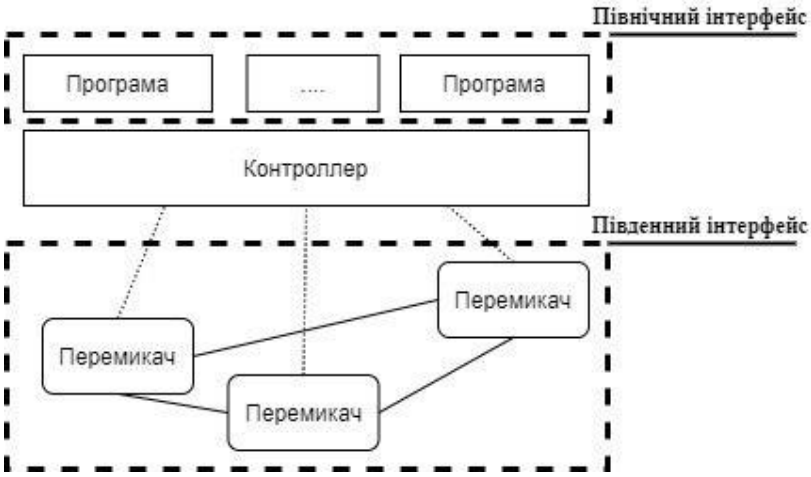


Рис. 1.1 – Структура SDN концепції

SDN-контролер є проміжним програмним забезпечення, яке діє на базі алгоритмів, які визначають маршрути пакетів. Алгоритми обчислюють маршрут пересилань пакетів, який і відправляється до таблиць мережевих пристроїв.

Контролер використовує протокол OpenFlow який є відповідальний за автоматичне налаштування обладнання. Протокол дозволяє відправляти повідомлення від контролера до перемикача, для того щоб конфігурувати перемикач, контролювати стан, керувати таблицями потоків. Протокол використовується для передачі пакетів ґрунтуючись на правила із таблиці потоків. Всі правила які є в таблиці - встановленні контролером. Протокол виконує роботу взаємодії між рівнем управління і базовою мережевою інфраструктурою.

Існують вимоги до OpenFlow маршрутизаторів, вони повинні складатися з одного або декількох таблиць потоків, групових таблиць і каналу до віддаленого контролера. Маршрутизатор обмінюється повідомленнями завдяки протоколу зображеному на (рис.1.2)[2].

Контролер використовуючи даний протокол може добавити, поновлювати та видаляти потокові записи в таблиці потоків. Таблиця

складається з полів повіряння, лічильниками, набором інструкцій. При попаданні пакету на маршрутизатор, він порівнює заголовок пакету з полями порівняння. Якщо знаходиться співпадаючий запис: то до пакету застосовують інструкції. Якщо ж не було знайдено запис, то задача делегується по тому як конфігурований маршрутизатор. Інструкції зберігають дії які необхідно застосувати до пакету і визначають самі правила пересилки. Є два варіанти обробки пакету, якщо не знайдено запис у таблиці, пакет інкапсулюється і відправляється контролеру, який формує відповідне правило для пакету даного типу і встановлює його на комутаторі, або пакет може бути відкинутий. Правила на комутаторі можна встановити реактивно, відповідаючи на прийнятий пакет, або про активно, тобто задати заздалегідь [2].

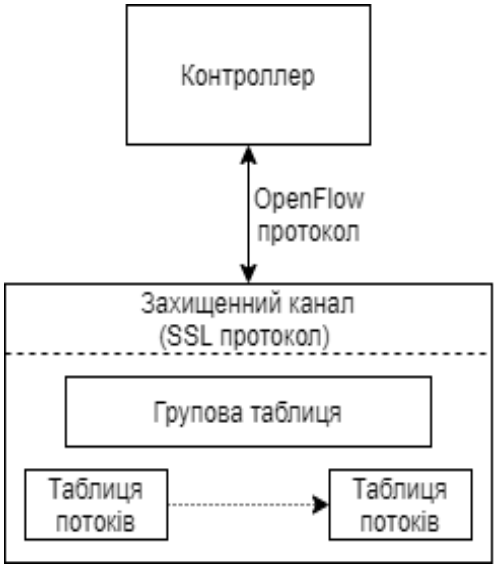


Рис. 1.2. Захищений протокол між контролером і маршрутизаторами

Маршрутизатори в традиційних мережах були фізичними приладами які знаходились в ЦОД, але за останні роки був розроблений рівень віртуалізації що надав можливість використовувати віртуальні маршрутизатори, прикладом такого для SDN мережі є Open vSwitch - це багаторівневий маршрутизатор, який поділений на 2 частини: користувацький простір і внутрішній простір. Цей маршрутизатор реалізує перемикач з таблицею потоків [3].

Перевагами SDN таким чином є оптимізація керування інфраструктурою ЦОД(центрів опрацювання даних), зменшення залежності від обладнання, масштабування, ефективність і гнучкість системи, швидкість реалізації потрібного функціоналу. Спрощує процес побудови маршрутів на відміну від традиційних мереж. Збільшує можливість ведення інновацій та дає можливість використовувати прості пристрої замість дорогих маршрутизаторів [2].

1.1.2 NFV підхід

Також для вирішення проблем традиційних мереж існує концепція віртуалізації мережевих функцій(NFV) - віртуалізація фізичних мережевих елементів, коли мережеві функції виконуються програмними модулями, які працюють на звичайних серверах. Данні програмні модулі взаємодіють між собою щоб надати послуги зв'язку, чим раніше займалися апаратні платформи, приклад на (рис.1.3) NFV дозволяє замінити дороге мережеве обладнання простими програмними приладами, збільшує швидкість розгортання за рахунок мінімізації інноваційного циклу мережевого оператора. Данна концепція потребує менше місця для розміщення мережевого обладнання, знижує енергозатрати, зменшує витрати на обслуговування мережі, дозволяє впроваджувати інновацію NFV направлена змінити те як мобільні оператори будують мережу, через розвиток базових технологій віртуалізації, щоб об'єднати більшість типів мережевого обладнання в індустріальний стандарт серверів. Розгортка віртуального обладнання може відбуватися за вимогою, ніякого нового обладнання не потрібно [4]. Хоча ця концепція з'явилась пізніше ніж програмно-конфігуруванні мережі, оператори розглядають можливість впровадження, так як це зв'язано з наступними особливостями не вимагає завчасно виводити із експлуатації обладнання яке вже працює, впровадження буде сприяти к скороченню витрат, впровадження може відбуватися поступово.

Рисунок 1.3. Структура NFV концепції

SDN і NFV можуть використовуватись окремо, але програмно-конфігуруванні мережі дають можливість простіше реалізувати і керувати віртуалізацією мережевих функцій [5]. SDN керує складність мережі через програну платформу, що забезпечує централізовану координацію і контроль. Без цього, NFV треба буде більше ручного втручання щоб налаштувати мережу.

1.2. Безпека систем мобільних мереж

Головним питанням таких систем є забезпечення безпеки в таких мережах. Так як програмно-конфігуруванні мережі надають більше можливостей для мобільних мереж, тому ця робота направлена на забезпечення безпеки для SDN концепції.

Архітектура SDN надає новий підхід для того щоб реалізувати мережеву інфраструктуру, але вона також вразлива з боку інформаційної безпеки. Існує декілька рівнів які є вразливими в даній системі. Можна виділити наступні: рівень програмних додатків, рівень контролю, рівень даних.

1.2.1 Рівні атак на SDN

Рівень програм [6]. Так як більшість мережевих функцій можуть бути реалізовані програмно в таких мережах і немає спеціалізованих стандартів для програм, які контролюють мережеві сервіси і функції через контролюючий рівень. Тому такі додатки створюють серйозні загрози безпеки мережевих ресурсів, служб та функцій. Хоча OpenFlow дозволяє модифікувати власні алгоритми з метою підвищення захисту. Тому існують на даний момент вже відомі реалізації безпеки, такі як:

1) автентифікація та авторизація - базова задача захисту інформації для забезпечення захисту від незаконного доступу до ресурсів з наданням певних прав та повноважень, авторизація .

2) контроль доступу та підзвітність – завдяки звітності проводиться аналіз по виявленню порушень, щоб обмежити тим часово доступ для можливих зловмисників.

Рівень контролю [6]. Найбільш вразливим компонентом в даній мережі є контролер, так як він є критичним елементом цієї системи, атака на цей елемент буде критичною для всієї системи. Тому можна виділити так атаки на контролер:

1) Загроза від додатків – реалізовані додатки поверх рівня контролю, можуть нести серйозні проблеми.

2) Загрози через масштабування – так як головну роботу виконує один елемент системи – контролер, якому може бути назначено встановити потокові правила для нового шляху наприклад. В цьому випадку контролер може стати тип елементом системи від якого залежить швидкість усієї мережі. На сьогоднішній день майже всі реалізації контролера не здатні опрацьовувати велику кількість нових правил. Через це система становиться вразливою до DoS і DDoS атак. Для того щоб уникнути цієї атаки, можливо використовувати кластер з SDN контролерів, тим самим забезпечити відмовостійкість. Але такий підхід тим те не менш може все одно спричинити каскадну відмову. Так

як все навантаження контролера який відмовив беруть працездатні контролери.

3) DoS атаки – DoS і DDoS або “Відмова в обслуговуванні” ця ситуація може виникнути на основі самого алгоритму роботи контролера, коли отриманий невідомий пакету який відсутній в таблиці потоків. При цій ситуації можливі два варіанти розвитку:

- Весь пакет пересилається до контролера щоб проаналізувати його
- Пакет залишається в пам’яті комутатора, на контролер відправляється лише заголовок пакета.

  Обидва методи є ефективним способом здійснити атаку. Контролер змушений опрацьовувати при таких атаках шкідливі пакети замість легітимних, що може знизити швидкодію мережі. Відбувається навантаження на канал який з’єднує контролер і атакований комутатор. Комутатор приймає команди від контролера і виконує їх, тим самим витрачаючи ресурси процесора і пам’яті. Якщо команди які прийшли мають в собі створення нових правил до таблиці, то відбувається збільшення часу на перевірку кожного пакету.

4) IP підробка - це атака яку використовують підчас DDOS атаках. Використовуються підроблені IP адреси щоб направити пакети в SDN мережу. Зазвичай виникає дана проблема при неправильній конфігурації мережі.

Рівень даних [6]. В мережі можливі атаки на протокол OpenFlow, один з відомих:

1) Атака на режими роботи контролера(реактивний і про активний). Режим роботи може бути легко визначений зловмисником ґрунтуючись на затримці для першого пакету нового трафіку. В результаті можуть використовуватись конкретні режими для проведення атаки. При несанкціонованому встановленні правил обробки на комутаторах, може призвести до зниження ефективності або перебою роботи мережі. Легше

реалізувати в реактивному режимі через підхід контролера до керування таблицями потоків в цьому режимі.

2) MITM - “людина посередні” це атака легка в SDN мережі через те що заголовок з автентифікацією відсутній у пакетів OpenFlow, що проходять через комутатор. Якщо атакуюча сторона перехоплює данні між контролером і комутатором, то зловмисник може продублювати трафік, і відправити свої правила до таблиці комутатора, щоб маті повний доступ до нього і пере направити весь трафік собі. Дана атака може вплинути на продуктивність і довіру до мережі.

3) Повторна атака - атаки що відбуваються через те що деякі пакети не зафіксовані часом, бо пакети надіслані “чистими” через канал. Це дозволяє атакерам перехвачувати пакети, та повторно їх пересилати, або затримувати її надходження шахрайським шляхом.

4) Модифікація інформації - відбувається під час передачі пакетів через мережу SDN. Так як пакети не шифруються та не підтверджуються автентифікацією, пакети даних надсилаються без будь-якого захисту. Це призводить до фальсифікації пакетів даних зловмисниками.

5) Головні загрози на різних рівнях SDN зображенні в таблицях 1.1, 1.2, 1.3

Таблиця 1.1. Загрози на рівні програм в SDN мережі

SDN рівень	Тип загрози	Опис
Рівень програм	Відсутність автентифікації та авторизації	Відсутність стандартів автентифікації та авторизації для великої кількості вже готових програмних модулів.
	Шахрайське вставлення потокових правил	Шахрайські додатки можуть генерувати хибні потокові правила.

Таблиця 1.1

SDN рівень	Тип загрози	Опис
Рівень програм	Відсутність контролю доступу та підзвітності	Складнощі впровадження контролю доступу та підзвітності в сторонніх модулях які поставляють мережеві ресурси

Таблиця 1.2. Загрози на рівні контролю в SDN мережі

SDN рівень	Тип загрози	Опис
Рівень контролю	Атаки «Відмова в обслуговуванні»	Централізація управління обмежує можливості контролера, що і сприяє DoS атакам.
	Неавторизований доступ до контролера	Відсутність механізму забезпечення доступу до додатків.
	Неавторизований доступ до контролера	Відсутність механізму забезпечення доступу до додатків.

Таблиця 1.3. Загрози на рівні даний в SDN мережі

SDN рівень	Тип загрози	Опис
Рівень даних	Шахрайські потокові правила	Рівень даних більше вразливий до шахрайських правил.
	Надмірна кількість правил	Потокові таблиці перемикачів можуть зберігати обмежену кількість правил, тому ця вразливість може використовуватись.

Таблиця 1.3.

SDN рівень	Тип загрози	Опис
Рівень даних	Захват контролера	Рівень даних залежить від рівня контрою, який в той самий час робить рівень даних залежним від безпеки самого контролера.
	Рівень TCP-атак	Протокол TLS сприятливий до TCP-атак
	MITM - “людина посередні”	Через те що не завжди використовують TLS і через його складність налаштування

Найбільш розповсюдженою проблемами які можна виділити є проблема автентифікації з авторизацією та шахрайські потокові правила.

1.2.2 Підхід до вирішення проблеми авторизації та автентифікації

Для того щоб запобігти більшості атак, які можливі через відсутність автентифікації, використовують IPsec tunnel с ESP заголовками. Взаємодія між OpenFlow комутатором і контролером зашифрована симетричним ключем, використовується алгоритм AES. Так як це симетричне шифрування, то контролер і всі комутатори мають один і той самий ключ. Контролер відправляє цей ключ до всіх перемикачів через протокол захисту транспортного рівня (TLS), який в свою чергу працює на асиметричному алгоритмі RSA. Вся комунікація між контролером і перемикачами шифрується. Одночасно це і створює проблеми для мережі, так як шифрування вимагає додаткових ресурсів, тому система стає більш вразливою до DDOS атак. Приклад роботи що описується на рис. 1.4 [7].

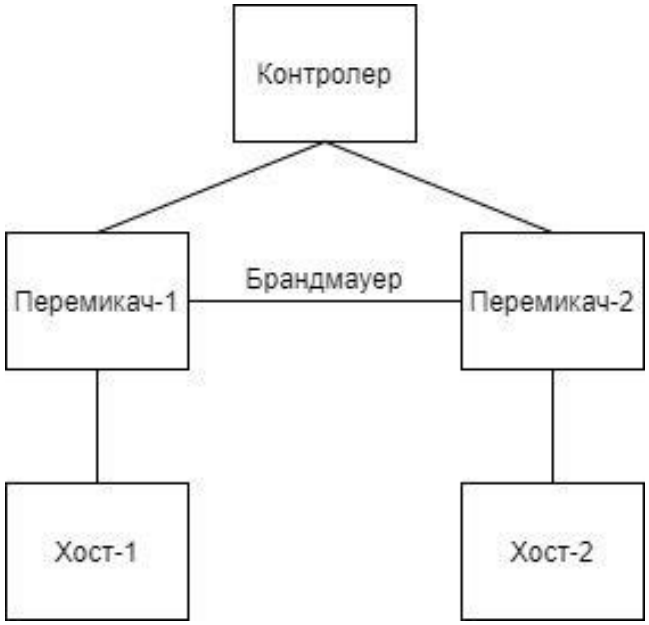


Рисунок 1.4. Приклад системи в якій використовується авторизація та автентифікація

Брандмауер побудований на інтерфейсах шлюзу на Перемикач-1 і Перемикач-2, через які підключені Хост-1 та Хост-2. Використовуючи топологію на рис. 1.4, коли Хост-1 захоче встановити з'єднання з Хост-2, він виправить повідомлення з спільним ключем і адресом шлюзу брандмауера Хост-2. Так як в таблиці потоків немає запису для цього потоку, то перемикач відправить пакет до контролера щоб обновити таблицю. Контролер відповість, і запис потоку обновлюється в двох шлюзах комутаторів. Контролер використовує повідомлення, щоб передати пакет до Хост-2. Він утворює зашифрований канал між Перемикач-1 і Перемикач-2. Шлюз брандмауера забезпечений списком доступних IP.-адрес Хост'а-1, для того щоб брандмауер на Перемикач-2 дозволив весь трафік від Хост-1. Далі передається пакети даних від Хост-1 до Хост-2, через зашифрований заголовок. Структура пакету IPsec Tunnel mode с ESP заголовком відображена на рис.1.5 [7].

Рисунок 1.5. Структура ESP заголовку

Даний підхід є легким в реалізації і має малий вплив на ресурси системи.

1.2.3 Підходи до вирішення проблеми шахрайських правил

Проблема шахрайських правил є більш складною. На даний момент існують декілька реалізацій які наведено в таблиці 1.4.

Таблиця 1.4. Реалізації усунення проблеми шахрайських правил

SDN рівень	Реалізація	Тип вирішення проблеми
Рівень програм	Assertion	Вирішення через відладка програм
Рівень даних	FortNox	Інструмент для контролера,
	FlowChecker	Інструмент для верифікації конфігурацій
	VeriFlow	Інструмент відладки мережі.
	BlockChain	Інструмент верифікації команд

Assertion – використовує VeriFlow, щоб розробники могли перевіряти динамічні значення програм контролера [8].

FortNox - розроблений алгоритм аналізу, щоб знаходити конфлікти правил в таблицях. Конфлікт виникає коли нове правило суперечить існуючим правилам. Алгоритм може вирішити чи приймати нове правило чи ні, залежно від того, чи той хто запросив вставку працює з більш високою авторизацією безпеки, ніж у авторів існуючих суперечних правил. Реалізує автентифікацію на основі ролі [9].

FlowChecker - аналізує всі конфігурації комутаторів, використовуючи бінарні рішення і модель перевірки щоб знайти конфлікти в таблицях потоків [10].

VeriFlow – перевіряє правильність мережі, виконує перевірку в реальному часі. Виконує моніторинг всієї мережі, далі перевіряються тільки ті випадки коли правила змінюються в таблицях і в кінці перевіряються інваріанти [11].

BlockChain(блокчейн) – збирає файли з логами з перемикачів та порівнює їх з командами які надсилав контролер. Дуже легкий спосіб аналізу, який не потребує великих ресурсів [12].

Серед всіх можливих реалізацій, BlockChain підхід не має реальної реалізації і в той же час він не потребує великих ресурсних затрат. Тому в даній роботі буде проводитись аналіз даного підходу.

1.3 Блокчейн як підхід до забезпечення безпеки в SDN мережі

Рисунок 1.5. Структура блоку в блокчейн мережі

Блокчейн технологія може бути ефективно використана для забезпечення безпеки в мобільних мережах. Головною ідеєю є децентралізація, це значить, що для атаки на цю систему треба отримати доступ до більше ніж 50% учасників мережі. Система являє собою просту структуру даних, щось на подобі однозв'язного списку. Основним компонентом цього ланцюгу є блоки (рис.5). Кожен блок складається з заголовка і самих даних які він зберігає. Структура заголовку зазвичай є такою: ідентифікатор блока, час створення, корінь Меркла та хеш значення поперечного блока. Винятком є зароджуючи блок, він не має попереднього блоку. Данні блока – це список транзакцій, тому

корінь меркла який необхідний для заголовку утворюється шляхом отримання хешу від комбінацій хеш-значень кожної з транзакцій [13].

1.3.1 Переваги підходу

Можна виділити наступні переваги блокчейну:

1) Децентралізація – не має стороннього серверу який верифікує транзакції, замість цього, верифікація відбувається завдяки процесу консенсусу [15].

2) Прозорість і довіра – так як в мережі блокчейну всі данні доступні кожному учаснику цієї системи, кожен може перевірити правдивість записаної інформації [15].

3) Незмінність – одного разу записана інформація до блокчейну, не може бути модифікована. Можливо тільки повна її заміна, але через складність алгоритмів шифрування, проблема підміни зникає [15].

4) Великий рівень доступності – так як блокчейн це P2P система, то кожному учаснику мережі досить того щоб хоча б ще один вузол працював в цій мережі щоб синхронізувати данні.

5) Великий рівень безпеки – всі транзакції в блокчейні захищенні криптографічним чином, тим самим надається цілісність всієї мережі.

6) Збереження витрат – так як будь-який учасник мережі може стати також учасником мережі блокчейн, немає потреби використовувати централізований сервер верифікації.

7) Відмово стійкість – так як кількість учасників не обмежена, це означає що система відмовить тільки тоді коли не буде хоча б одного учасника в мережі, що майже не можливо.

1.3.2 Консенсус як ключовий елемент блокчейну

Процес консенсусу – це процес який направлений щоб визначити блок який буде добавлений в ланцюг і яким є стан системи на даний момент. Існує багато алгоритмів консенсусу, такі як:

1) Proof of Work (доказ роботи) – отримав свою назву через те що для того щоб визначити блок треба підібрати спочатку потрібне хеш значення, яке всі учасники мережі намагаються знайти перебором. Є не надто ефективним, так як комп'ютерні потужності не виконують в цей час ніякої продуктивної роботи. Складність хешу який потрібно знайти контролюється самою системою [14].

2) Proof of Stack (доказ ставки) – Алгоритм базується на ідеї того що вузол або користувач має достатню ставку в системі, тим самим вважається що даний користувач є достатньо надійним, так як вигоду яку він може отримати нечесним шляхом є менше від його кредиту довіри [14].

3) Delegated Proof of Stack (делегуючий доказ ставки) – це інноваційний підхід на основі Proof of Stack, при цьому вузол який має більшу ставку може делегувати верифікацію блоку іншому вузлу через голосування [14].

4) Proof of Elapsed Time (доказ пройденого часу) – призначення його в тому щоб рандомізувати вибір через голосування вузла. Час гарантує правильність [14].

5) Proof of Deposit (доказ через депозит) – в цьому випадку вузли які бажають прийняти участь в мережі, спочатку повинні зробити безпечний депозит, перед тим як почнуть створювати і пропонувати блоки [14].

6) Proof of Importance (доказ через значимість) - дана ідея протилежна Proof of Stack. Консенсус полагається не тільки на те на скільки велика ставка, але й спостерігати використання і пересування токенів користувачами для того щоб встановити певний рівень довіри [14].

7) Reputation-based mechanism (механізм оснований на репутації) – головний учасник голосування визначений за репутацією яку він отримав за певний час знаходження в мережі. Через голосування лідер визначає правильність блока [14].

8) Proof of Activity (доказ активності) – це об’єднання Proof of Stack і Proof of Work, механізм забезпечує те що власник ставки вибраний псевдовипадковим чином [14].

9) Proof of Capacity (доказ кількості) – це схема використовує дисковий простір як ресурс для добування блоків [14].

10) Proof of Authority (доказ авторитету) – схема за якою головний вузол визначається шляхом голосування учасників мережі. Використовуються певні алгоритми для досягнення цієї мети [14].

1.3.3 Питання безпеки самого блокчейну

Як можна помітити, блокчейн використовує криптографічні алгоритми для отримання хеш значень. Хеш значення – це перетворений вхідний масив даних довільної величини в строку фіксованої довжини, з неможливістю отримати вхідні дані. При цьому є важливим наявність у алгоритму утворення хешу лавинного ефекту, при якому зміна хоча б одного значення з вхідного масиву потребує зміни половини вихідного значення. Проблемою є колізія хеш функцій. Також блокчейн використовує асиметричний метод криптографії на основі приватного та публічного ключа для цифрового підпису. Криптографія вносить в цю систему конфіденційність кожного вузла, цілісність даних, автентифікацію, неможливість відмінити транзакцію.

1. Конфіденційність означає те що тільки авторизовані учасники мережі можуть розуміти повідомлення один одного.

2. Цілісність даних означає що данні не можуть бути підроблені або модифіковані, випадково або з наміром. Але цілісність даних не означає, що їх не можливо замінити на альтернативні, цілісність дозволяє уникнути тільки модифікації даних.

3. Автентифікація – справжність відправника підтверджується та перевіряється тим хто одержує повідомлення.

4. Неможливість відмінити транзакцію – означає те що відправник, після відправки повідомлення не може відмінити його пізніше. Тобто учасник мережі не може відмовитись від транзакцій які він виконав.

1.3.4 Види організації блокчейн мережі

Ця технологія має різні типи реалізації такі як: блокчейн консорціуму, публічний і приватний блокчейн [16].

Блокчейн консорціуму – це розподілена книга в якій процес консенсусу контролюється наперед заданим набором вузлів. Зчитування з такого ланцюгу даних можуть виконувати самі вузли які перевіряють блоки або авторизовані наглядачі. Така система є частково децентралізованою.

Публічний блокчейн – це тип який був перший реалізований для фінансових послуг. Кожен учасник цієї мережі має повний доступ та може виконувати транзакції. Всі учасники можуть вносити свою частку в процес консенсусу.

Приватний блокчейн – доступ до такої мережі мають тільки обмежений список вузлів.

Висновок до розділу 1

Проаналізували проблеми традиційних мобільних мереж, визначили основні напрями розвитку для покращення швидкодії цих мереж. Визначили основні недоліки зв’язані з безпекою. Розглянули основні атаки, виділили основні. Розглянули існуючі рішення по забезпеченню безпеки в SDN мережі. Запропонували новий підхід на основі блокчейну. Проаналізували основні компоненті для побудові мереж блокчейн. Розглянули типи мережі блокчейн.

РОЗДІЛ 2
РОЗРОБКА БЛОКЧЕЙН МЕРЕЖІ

2.1 Розробка алгоритму консенсусу

Алгоритм консенсусу є ключовим елементом мережі блокчейн, завдяки ньому визначаються вузли які мають право добавляти новий блок до ланцюгу, так щоб всі вузли знали від кого очікувати ці данні. Вибір алгоритму напряму залежить від типу блокчейну який буде працювати. Для захисту SDN потрібен приватний ланцюг. Для даного типу більш всього підходять такі алгоритми консенсусу:

1) Proof of authority – транзакції і блоки верифікуються довіреними користувачами.

2) Proof of elapsed time – той хто буде з користувачів добавляти блок в ланцюг визначається завдяки часу.

3) Proof of Importance – визначає того хто добавляє блок в ланцюг через кількість довіри.

Інші алгоритми наприклад Proof of Work, Delegated Proof of Stack більше підходять для публічних реалізацій, так як там відбувається гонка. Дуже великі відбуваються затрати ресурсів наприклад в Proof of Work, він виконує насправді не потрібну роботу, тим самим витрачає ресурси марно.

Proof of elapsed time не має конкретних обмежень до тих хто може добавляти блоки, а Proof of Importance робить систему більше централізованою тому для цієї роботи вибираємо алгоритм Proof of Authority – який має не складну реалізацію. Алгоритм базується на N кількості вузлів які він називає авторитетами. Кожен вузол має унікальний ідентифікатор, більшість з них признаються правдивими. Час ділиться на шаги, щоб кожен учасник міг мати однакові права. Він має 2 підходи до реалізації насправді:

Aura – новий блок пропонує вибраний лідер. Даний підхід вимагає раунд прийняття блоку, для Clique це не потрібно. Будем вважати, що кожен учасник мережі працює в рамках одного і того самого UNIX часу T. Індекс K

кожного кроку визначається кожним клієнтом самостійно. Формула, що демонструє визначення кроку, де інтервал це константа.

$$К = \frac{Т}{інтервал} \quad (2.1)$$

де **К** – індекс кроку;

Т – UNIX час системи;

Лідер Л кожного шагу визначається за формулою:

$$Л = К \ mod \ N \quad (2.2)$$

де **Л** – лідер кожного шагу;

К – індекс кроку;

N – кількість учасників мережі блокчейн:

Авторитет підтримує дві черги, одна з транзакціями «Черга транзакцій», а друга з блоками «Черга блоків». Кожна отримана транзакція зберігається в «Черга транзакцій». Кожного кроку, лідер включає всі транзакції з «Черга транзакцій» в блок, який і розміщує в «Черга блоків». Далі транслює чергу до інших учасників мережі. Далі кожний учасник мережі відправляє отриманий блок до інших учасників, вони приймають цей блок і вносять в «Чергу блоків». Будь-який отриманий блок від учасника який не є лідером на даний момент відхиляється. Лідер завжди повинен відправити блок, якщо немає транзакцій в «Черга транзакцій» він вимушений відправити пустий блок.

Якщо авторитети не погоджуються з отриманим блоком за раунд прийняття, то проходить проінформування адміністратора мережі, щоб він міг втрутитись та перевірити стан сисстеми.[17].

Clique – новий блок пропонує вибраний лідер. Лідер і шаг визначається за формулою яка використовує номер блоку і номер авторитету. Більше того, поточному лідеру і іншим учасникам дозволено пропонувати блок на кожному кроці. Щоб уникнути ситуації коли один учасник добавив велику кількість блоків, кожному авторитету дозволяється пропонувати блок кожні N/2 + 1 блоки. Виходить в один момент часу дозвіл мають пропонувати N – (N/2 + 1) блоки [16].

Рисунки 2.1 та 2.2 демонструють принцип роботи Clique та Aura. На рисунках відображено 4 користувача з ідентифікаторами 0, 1, 2, 3.

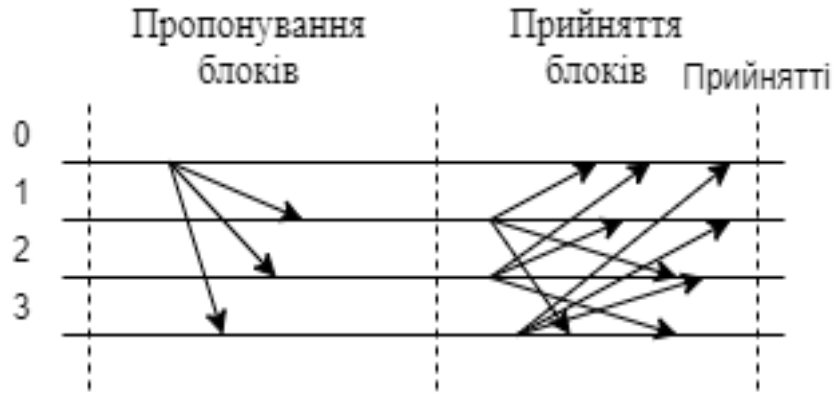


Рис. 2.1. Aura реалізація.

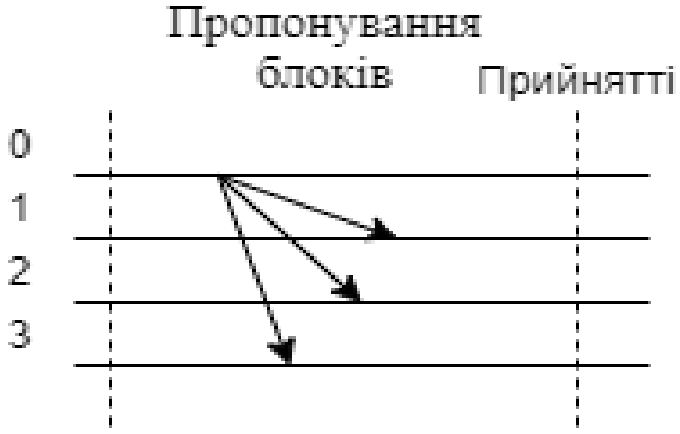


Рис. 2.2. Clique реалізація.

Як реалізація для цієї роботи буде вибрана Aura реалізація.

2.2 Аналіз протоколу взаємодії вузлів

Блокечейн можна також назвати розподіленою базою даних, так як в ньому зберігаються данні. Ці дані зберігає кожен з учасників мережі, але для того щоб кожен учасник міг пересилати данні до іншого користувача потрібен мережевий протокол. Задача користувача – передати набір текстових даних, іноді структур даних які містять ці данні. Свій вибір будем робити з точки зору ефективності, тобто щоб протокол був низького рівня, щоб зменшити кількість спеціальних заголовків. Для реалізації комунікації пропонується використати один з наступник протоколів: UDP, TCP які відносяться до транспортного рівня моделі OSI, та FTP, HTTP які відносяться до прикладного рівня. Зазначу що, для даної мережі буде використовуватись тільки оперативна пам’ять

комп’ютера, твердотільні накопичувачі не будуть використовуватись. Використання тільки оперативної пам’яті повинно надати максимальну швидкодію кожному з вузлів.

2.2.1 Порівняння протоколів

UDP - протокол транспортного рівня, у своєму заголовку містить інформацію про порт відправника і контрольну суму. Порт відправника дозволяє ідентифікувати відправника, а контрольна сума правильність отриманих даних. Протокол не гарантує що данні будуть отриманні. Так як протокол не гарантує передачу даних, то виправлення помилок або повторна передача даних можливі, данні фактори можуть впливати на швидкодію всієї мережі. Так як в цьому протоколі відбувається виправлення помилок, то саме повідомлення розширюється надлишковим кодом. Заголовок займає 8 байт.

TCP – протокол транспортного рівня, працює з потоками даних, розбиває данні і кожній частині назначає заголовок з номер в послідовності. Частини далі інкапсулюються в IP-пакет, а далі передається по IP протоколу до користувача. Користувач перевіряє коректність отриманих даних, після чого повідомляє відправника що всі данні були отриманні коректно, або потребує повторної передачі. Заголовок містить порт джерела та порт призначення, номери послідовностей сегментів, самі данні та багато ще спеціалізованих значень. Протокол забезпечує надійну передачу даних. Розмір заголовка з міняльною кількістю інформації займає 20 байт, але якщо використати всі опції то займе 60 байт.

FTP – протокол прикладного рівня, дозволяє користувачам обмінюватись двійковими або текстовими файлами. Використовує два TCP з’єднання між клієнтами. Має ті самі переваги, що і TCP, так як гарантує передачу даних. Може працювати в двох режимах: активному і пасивному. При активній роботі один користувач передає данні і потребує щоб їх прийняли, а в пасивному віддає данні коли йому приходить запит на конкретні ресурси. Згідно з специфікації RFC, має заголовок який складається з дескриптора розміром 8

біт та 16 біт які відображають кількість байт які передають. Також використовується компресія даних при передачі.

HTTP – протокол прикладного рівня, широко використовується в клієнт-серверній схемі. Оснований на TCP протоколі, тому забезпечує передачу даних, Запит і відповідь мають різну структуру трохи, Частина інформації в заголовку є обов’язковою, що може займати певний розмір даних.

FTP і HTTP – протоколи високого рівня, дозволять робити більш гнучку взаємодію між користувачами, але то же самий час містять додаткову інформацію, яка є зайвою для мережі блокчейн. Дані протоколи більше спеціалізовані для клієнт-серверної архітектури. У випадку с блокчейном потрібна P2P мережа, тобто архітектура взаємодії клієнт-клієнт, так як мережа націлена на децентралізацію.

Тому вибір зупиняється на TCP і UDP протоколах. З точки зору передачі даних в блокчейні, можна зробити висновок що передача даних, не є обов'язковою, вона повинна відбуватись, але підтвердження передачі не є значимим. TCP вносить надлишкові затрати на передачу даних для того щоб досягти їх цілісності, тому даний протокол хоча є дуже надійним, але не підходить для даної системи. Оптимальним протоколом який і буде використовуватись в мережі блокчейн буде UDP.

2.2.2 Порівняння результатів тестів

Для повної впевненості проведемо тести порівняння TCP і UDP. Порівнювати їх з протоколами прикладного рівня немає сенсу, так як протоколи високого рівня гуртуються на протоколах прикладного рівня. Визначати будем на одній і тій самій машині, передавати будем один байт даних. Вимірювати будем в мега-бітах за секунду. Результати наведені на (рис.2.3) [18]. Результат є очевидним, так як TCP має накладні розходи на підтвердження отримання даних. Варто зазначити що UDP втратив 80% всіх відправлених даних. Але як зазначалось раніше, блокчейн не потребує 100% передачі даних.

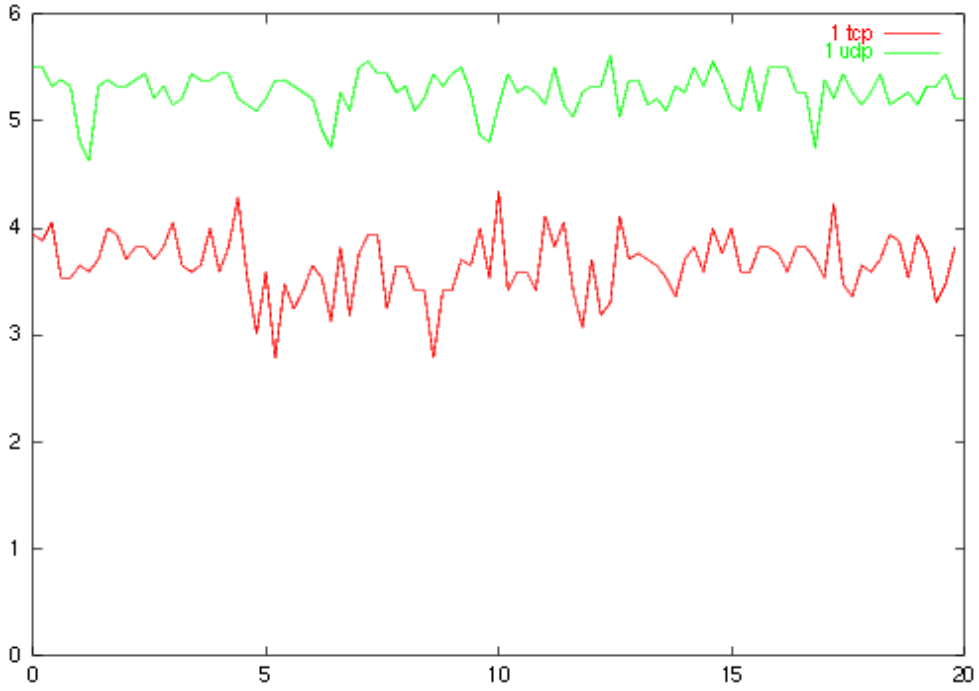


Рис. 2.3. Графіки передачі мега-біт за секунду протоколів UDP та TCP. Червоний колір відображає TCP, зелений UDP

2.3 Аналіз криптографічних хеш-функцій.

Криптографічні алгоритми є важливою складовою мережі блокчейн. Вони дозволяють перевіряти данні, так як для кожного блок в ланцюгу визначений хеш. Тому при спробі змінити данні в блокчейні, зловмиснику потрібно перерахувати всі хеш значення. Дана робота потребує велику кількість ресурсів і виграш який може отримати зловмисник є меншим за затрати на ресурси.

2.3.1 Хеш-функції

Хеш-функції – функція яка перетворює довільний розмір даних у строку фіксованої довжини. Хеш функції гарантують незворотність, тобто неможливо з хеш-значення отримати початкові данні [19]. Вимоги до хеш-функції є такими:

1) Забезпечення захисту від колізій

2) Володіє лавинним ефектом – тобто при зміні хоча б одного біту, змінюються вихідне значення на половину.

Головною проблемою хеш-функцій є боротьба с ASIC приладами. ASIC – спеціалізоване обладнання для визначення хеш-значення для конкретного алгоритму. Їхня потужність на декілька порядків вища ніж процесора комп'ютера або відеокарт. Через їх велику потужність вони можуть перераховувати велику кількість хеш значення, тим самим добавляти або видаляти транзакції які будуть для зловмисників вигідними. Тому в даній роботі буде проведений аналіз алгоритмів і вибраний такий для якого не вигідно створювати дане обладнання буде.

Варто зазначити, що розглядатись будуть тільки криптографічні хеш-функції.

MD5 – хеш-функція яка в результаті своєї роботи видає 128-бітні хеш значення. Було знайдено вразливість при якій дана функція страждає, також вона піддається перебору. Атака колізії присутня, відомо що значення колізій знаходяться за лічені секунди. Хоча дана система все ще застосовується, але вона не є хорошим вибором.

SHA256 – криптографічна хеш-функція в результаті роботи видає хеш-значення розміром 256 біт. Данна функція широко застосовується в багатьох блокчейн мережах. Хоча криптографічних вразливостей на початку створення алгоритму не було знайдено, але з часом знайшли колізії для 22 ітерації та для 31 та атаки додавання повідомлення. Також створено багато ASIC приладів для знаходження цього алгоритму.

X11 – криптографічний алгоритм який використовує послідовність з 11 алгоритмів хешування. Алгоритм був одним із найбезпечніших алгоритмів та був енергоефективним для системи до того як в 2016 році були розроблені ASIC прилади для нього.

SHA-3 – криптографічний алгоритм шифрування в якому відсутні всі недоліки SHA256 так як він має іншу структуру. На даний момент ASIC прилади для даного алгоритму відсутні. Результат хеш-функції є вижимка криптографічної губки. Можливі розміри хеш функції 224, 256 384, 512 біт.

З перечислених хеш-функцій буде використовуватись в цій роботі новий стандарт тобто SHA-3 з 256 бітним хешем.

2.3.2 Дерево Меркла

Кожен блок блокчейну зберігає список транзакцій, як тільки блок становиться частиною блокчейну він є незмінним. Це означає, що якщо транзакція присутня в одному блоці, то вона не буде присутня в жодному іншому блоці блокчейну. Транзакції перераховані в дерево меркла або бінарне-хеш дерево.

Корінь дерева знаходиться з верху, вузли які знаходяться знизу називаються листями. Кожен вузол просто є криптографічним хеш-значенням транзакції. Дерево меркла не зберігає список всіх транзакцій, воно зберігає хеш всіх транзакцій у вигляді дерева [20].

$$\text{Припустимо: хеш алгоритм} = SHA256 \quad (2.3)$$

$$\text{Хеш транзакції } 0 = \text{Хеш}\big(\text{Тр}(0)\big) = SHA256(SHA256(\text{Транзакція A})) \quad (2.4)$$

Кожен хеш обраховується завдяки використанню хеш алгоритму двічі.

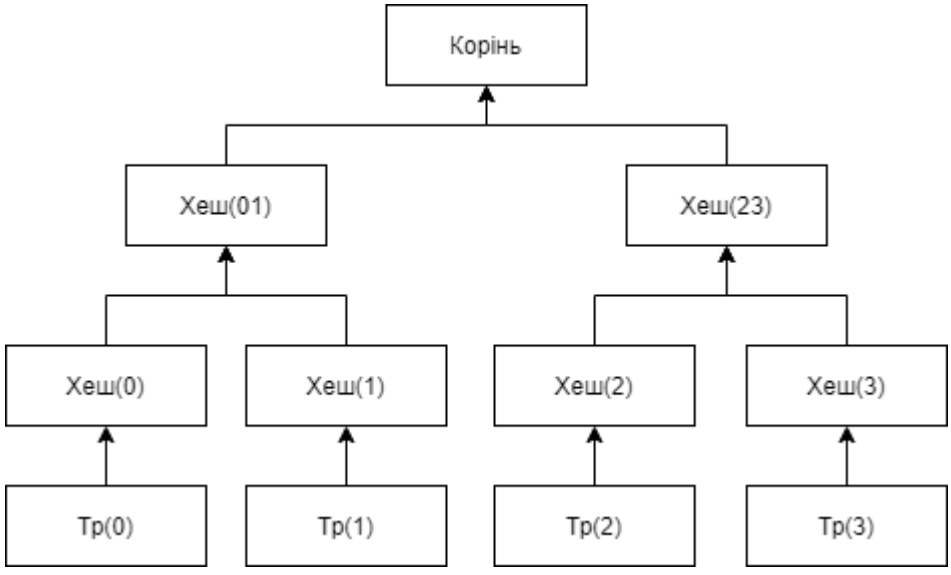


Рис. 2.4. Дерево Меркла

2.4 Аналіз криптографічних алгоритмів передачі даних

Мережа блокчейн в даній роботі буде базуватись на приватній мережі, тому комунікація користувачів буде тільки з тими хто є в списку довірених.

Але з’єднання між користувачами залишається не захищеним. Можливі атаки MIMT, коли пакет який пересилають, можуть зловмисники змінити при передачі. Для того щоб вирішити цю проблему можна використати алгоритм шифрування даних. Є три можливі варіанти роботи:

1) Використання асиметричного шифрування на основі приватних та публічних ключів.
2) Використання симетричного шифрування.
3) Використання цифрового підпису

2.4.1 Асиметричні алгоритми

Асиметричне шифрування базується на тому що кожен учасник мережі має свій закритий та відкритий ключ який знаходиться у кожного учасника мережі, тим самим коли вузол відправляє данні, він зашифровує данні публічним ключом того учасника до якого відправить данні. Для асиметричного шифрування існують найбільш популярні алгоритми, такі як:

RSA – асиметричний криптографічний алгоритм, алгоритм оснований на факторизації цілих чисел, тому є дуже ресурсовитратним. Взаємодія користувачів працює таким чином: кожен користувач має публічний ключ кожного учасника приватної мережі блокчейн, коли учасник мережі хоче передати повідомлення до іншого, то він бере публічний ключ того користувача до якого хоче відправити данні і шифрує їх цим ключем. Після чого користувач на іншій стороні отримує це повідомлення і відкриває його своїм приватним ключем. Так же алгоритм піддається атаці Padding Oracle, яка основана на вирівнюванні повідомлення.

ECC – алгоритм оснований на еліптичних кривих. Ключі використовуються значно меншого розміру ніж в RSA, але для нашої системи це не має великого значення. Принцип роботи ключів такий же як і в RSA. Має більш складний алгоритм, надійність залежить від формули кривої яка вибрана. Також не потребує вирівнювання тому він є стійким до Padding Oracle атаки.

2.4.2 Симетричні алгоритми

В даних алгоритмах використовується один ключ для шифрування і розшифрування. Швидкість цих алгоритмів значно більша ніж у асиметричних. Звичайною практикою мережевих протоколів є передача ключа для симетричного шифрування завдяки асиметричного шифрування. Так як велику кількість даних зашифрувати симетричним ключем буде значно швидше. Для симетричного шифрування існують найбільш популярні алгоритми, такі як:

1) RC4 – симетричний шифр, широко використовувався в транспортних протоколах, має високу швидкість роботи. Але були знайдені вразливості при використанні не випадкових або зв'язних ключів та при використанні ключового потоку двічі. Тому на даний момент є рекомендація не використовувати його.

2) DES – алгоритм симетричного шифрування, використовує блочне шифрування, кожен блок дорівнює 64 бітам інформації. В основі лежить мережа Фейстеля з 16 раундами. На крипостійкість даного алгоритму впливають S-блоки. Вони повинні видавати результат нелінійно. Також повнено бути лавинний ефект у кожного блоку, тобто при змінні одного біту повинно принаймні змінитись половина бітів на виході [19]. На алгоритм існують наступні види атак:

- Повного перебору – дуже велика кількість ітерацій, тому даний варіант ніколи не застосовують.
- Диференційний криптоаналіз – вона знизила на декілька порядків складність в порівнянні з повним перебором, але даний аналіз не завжди можна використати. Тому автори вважають що цей підхід не є доцільним.
- Лінійний криптоаналіз – дає постійний результат за константну кількість ітерацій.

3) AES - симетричний алгоритм шифрування, є новим стандартом, для того щоб замінити DES. Використовує ключі розміром 128 біт. Швидкість шифрування більша ніж у DES.

2.4.3 Цифровий підпис

Цифровий підпис дозволяє ідентифікувати користувача який підписав дані. Даний принцип ґрунтується на асиметричних алгоритмах. Але ключовим елементом цієї схеми є хеш, так як алгоритмом підписується тільки він. Тим самим економляться ресурси на шифрування и розшифрування. Важливим моментом є те, що потрібно підписувати хеш значення, це значення повинно бути отримано стійким хеш алгоритмом. Інакше повідомлення буде вразливе до атаки на колізії. Недоліком такого підходу залишаться те що кожен учасник мережі повинен мати публічний публічний ключ кожного учасника мережі.

2.4.4 Підсумок

Для мережі блокчейн важливим питанням є безпека передачі даних, для того щоб досягнути цього потрібно використовувати алгоритми шифрування. Важливим питанням є швидкодія мережі, тому асиметричні алгоритми не будуть хорошим вибором. Симетричні надають швидкодію і захищають данні надійно, але при викрадені всього одного цього ключа, система дискредитується. Тому найбільш вдалий підхід є використання цифрового підпису. Для системи яку ми конструюємо, кожен учасник мережі буде зберігати публічні ключі кожного учасника мережі. Асиметричний алгоритм шифрування який найбільше підходить для мережі є ECC, тому він і буде використовуватись.

2.5 Аналіз формату даних

Формат передачі даних є також одним із важливих питань, так як його формат може потребувати додаткового тексту який буде передаватись мережею, тим самим зменшить швидкодію програми. Також не правильний вибір може поставити під загрозу розвиток даної мережі. Розглянемо наступні формати даних: XML, JSON, серіалізація, CSV.

2.5.1 Аналіз XML, JSON, CSV, серіалізації

XML – мова розмітки яка використовується для передачі в мережі інтернет. Має ієрархічну структуру даних. Має придатний вигляд для читання людиною. Складається з тегів, всі теги починаються з відкриваючого і закінчуються закриваючим тегом. Через те потрібно слідкувати за тегами, робить ймовірність допустити помилку більшою, та потребує більше пам’яті так як величина тега залежить від його назви і він присутній двічі у документі. Також займає значний час на обробка документа для представлення його зручним для мов програмування.

JSON – текстовий формат для передачі в мережі інтернет. Основується на тексті який зрозумілий людині. Формат схожий на об’єкти в мовах програмування. Відсутні додаткові теги. Розмір значно менший в порівнянні з XML. Парсинг даних відбувається значно швидше ніж XML.

Серіалізація – процес перетворення об’єктів в масив бітів. З точки зору мов програмування є дуже зручним форматом для користування, при використанні сторонніх бібліотек можна досягнути дуже гарних результатів в перетворенні об’єктів в послідовність бітів і навпаки. Але даний підхід робить систему залежної від конкретної мови програмування або навіть версії окруження в якому виконується операції над цими даними. Це може вплинути на подальший розвиток мережі, що є не дуже гарним тоном в перспективі.

CSV – формат даних, націлений на відображення таблиць даних. Данні відокремлюються комами. При прямому використанні людиною, без сторонніх застосунків є дуже не зрозумілим для прочитання. Має гарну швидкість парсингу на рівні з JSON форматом.

2.5.2 Підсумок

З усіх варіантів оптимальним варіантом є JSON, так як він дозволяє вільно прочитати людині інформацію яку вона отримала. Перетворення цього формату в об’єкти в мовах програмування проходить дуже швидко.

2.6 Розробка структури даних блоків і ланцюжок

Формат даних визначений, тепер можна розробити структуру даних які будуть задіяні в проекті. Правильність вибору формату наддасть можливість на вдосконалення і розширення мережі у майбутньому.

2.6.1 Розробка структури транзакції

Транзакції – цей елемент системи блокчейн буде зберігати команду контролера, відповідний запис лог-інформації з перемикача та результат порівняння. Система буде постійно опитувати контролер всі перемикачі. Якщо команда на перемикачі буде такою, яка не збігається з жодною яку видав контролер буде сформована транзакція з пустою командою контролера і результатом атаки. Також буде зберігатись хеш значення на основі 3 значення цієї транзакції. Структура транзакції зображена на (рис. 2.5).

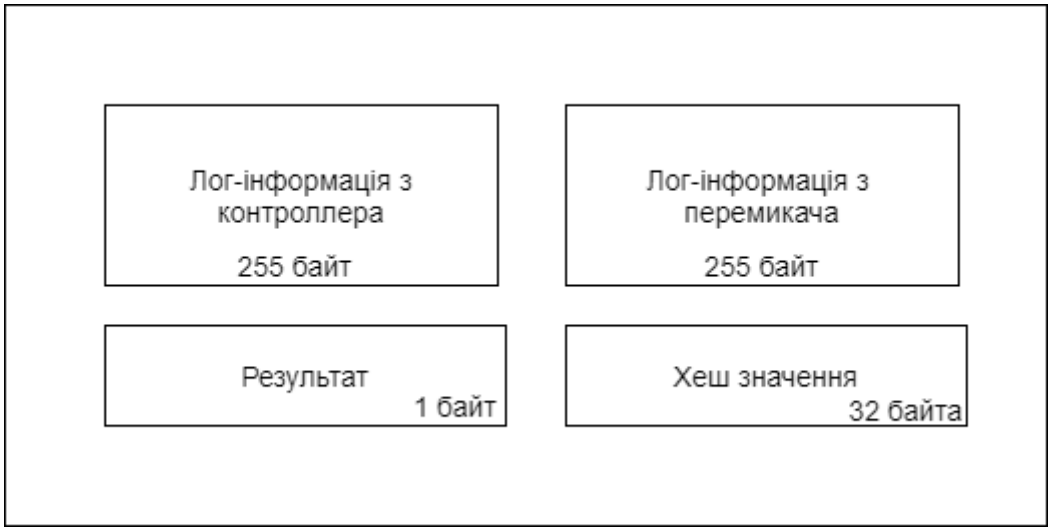


Рис. 2.5. Структура транзакції

Основна інформація в транзакції буде займати наступну кількість пам’яті:

Лог-інформація з контролера – буде обмежена 255 символами, тобто 255 байтам, або 2040 бітам, така кількість буде зарезервована для цієї інформації.

Лог-інформація з перемикача – буде обмежена також 255 символами, 2040 байтами.

Результат – буде булевим значенням, тому його розмір займе 1 байт даних, або 8 біт.

Хеш значення буде включати в себе всі 3 параметра транзакції, а результат буде займати відповідно до хеш-алгоритму 256 біт.

Максимальний розмір даних однієї транзакції не перевищить 4344 біти.

2.6.2 Розробка структури блоку

Блок – елемент системи який буде зберігати список транзакцій, корінь Меркла, час створення блоку, хеш-значення минулого блоку, ідентифікатор. Корінь меркла буде вираховуватись на основі транзакцій, хеш-значення минулого блоку. Кожен блок отримає унікальний ідентифікатор, для генерації ідентифікатора буде використаний uuid, який має розмір 128 біт, надає унікальні ідентифікатори, шляхом генерації випадкових послідовностей. Вірогідність колізії дуже мала. Буду використовуватись захищений генератор випадкових послідовностей, так як звичайній генератори базуються на рядах, тому їх результат можна підрахувати. Основна інформація в блоці буде займати наступну кількість пам’яті:

Корінь Меркла – буде дорівнювати вихідному значенню хеш функції, в нашому випадку для SHA-3 або Keccak[512] буде 256 біт.

Хеш-значення минулого блоку займе стільки ж пам’яті скільки і корінь Меркла.

Ідентифікатор – генерується завдяки uuid бібліотеки, значення буде дорівнювати 128 біт.

Час створення блоку мілісекундах – значення поміщається в формат long, тому змінна буде займати 8 байт, або 64 біти.

Розмір транзакцій буде необмежений.

Загальний розмір обов’язкової інформації в пустому блоці буде становити 896 біт, вважаючи що пустий масив транзакцій займає зазвичай 24 байти або 192 біти.

Рис. 2.6. Структура блоку

2.6.3 Розробка структури ланцюгу

Ланцюг буде списком блоків, які зберігають списки транзакцій. Ланцюг буду мати початковий блок, у якого буде відсутній хеш минулого блоку, через це корінь Меркла не буде враховувати це значення, список транзакцій буде також пустим відповідно.

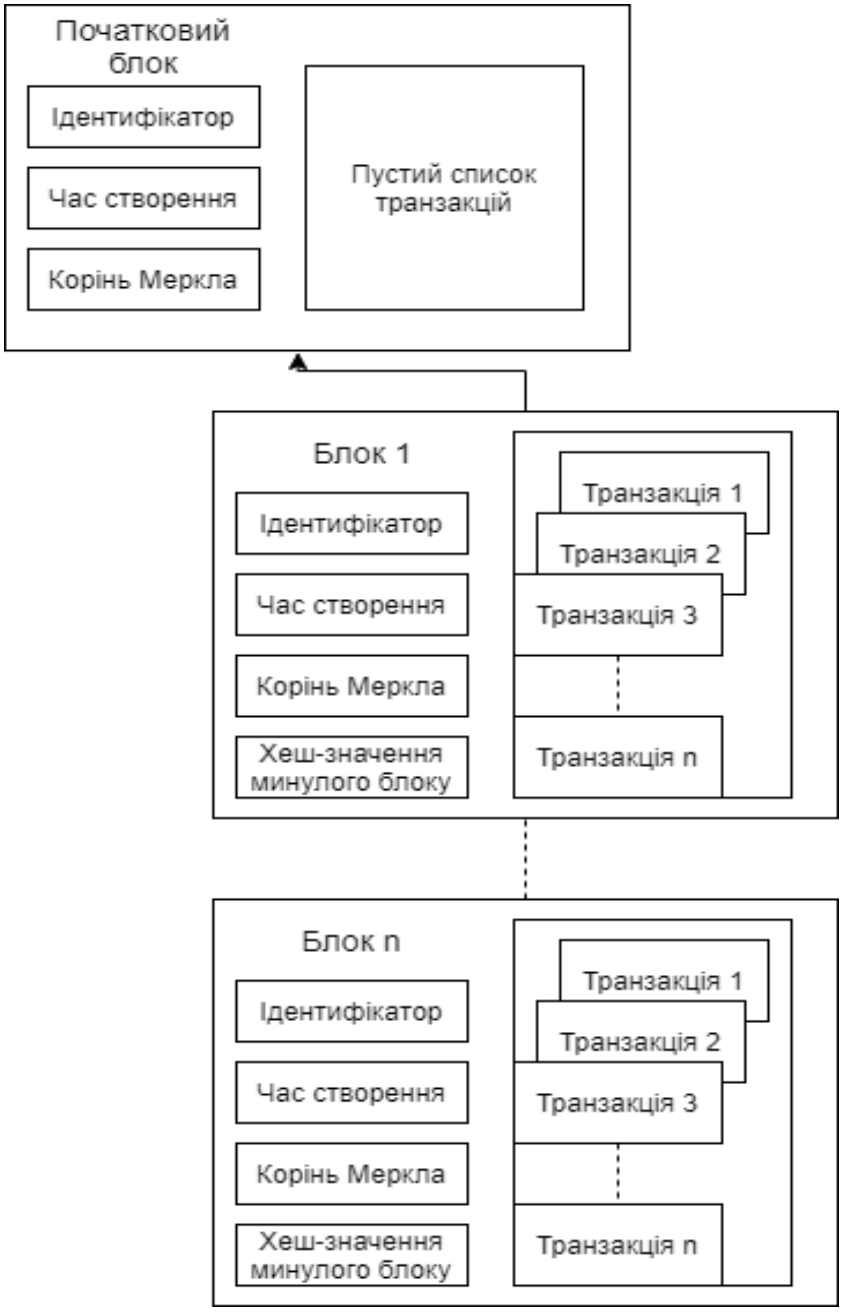


Рис. 2.7. Структура ланцюгу

2.7 Аналіз оптимальної кількості вузлів та системи сповіщення про загрози

Головним елементом цієї системи повинен бути сервіс який і буде сповіщати адміністратора про команди які з’явилися на перемикачі, але контролер їх не відправляв. Даний елемент системи буде називатись брандмауер. Для зручного моніторингу буде написаний HTTP сервер, який буде одночасно і учасником блокчейну і елементом моніторингу. Даний

учасник також буде приймати участь у створенні блоків. Схема буде повнозв’язною.

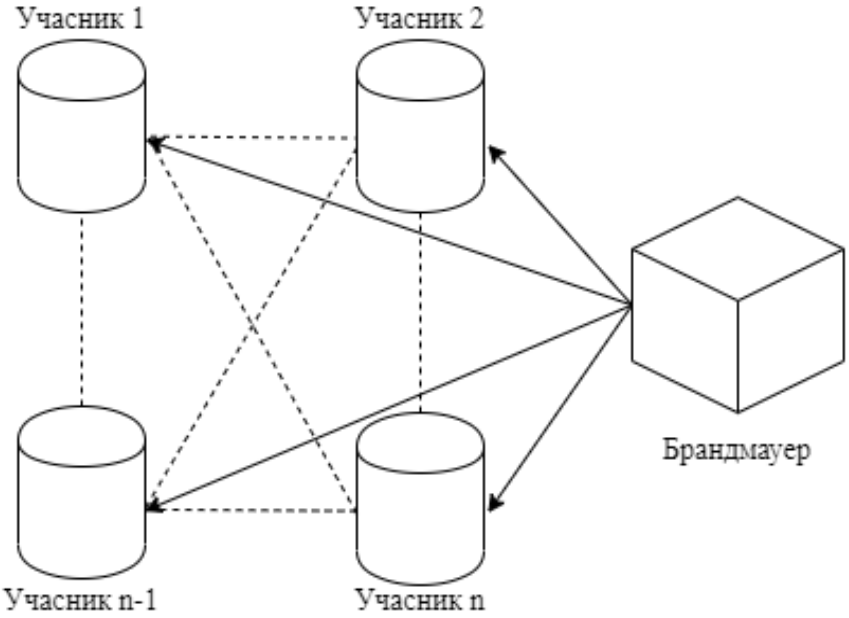


Рис. 2.8. Структура схеми роботи блокчейну з моніторинговою системою

Важливим залишається питання кількості учасників. Зі схеми зрозуміло, що вразливим елементом є Брандмауер. Тому для нього ми використаємо ненавантажене резервування. Визначимо відмовостійкість при 3, 4, 6, 8 учасниках мережі, відповідно при 1, 2 елементах резервування і без резервування у брандмауера. Для нашої задачі припустим що всі учасники і брандмауер мають вірогідність безвідмовної роботи за час Т дорівнюватиме 0.9. Вірогідність відмови буде розподілена за експоненційним законом.

Для розрахунку буде використовуватись метод згортки. Приклад розрахунку вірогідності без резервування:

Визначимо ймовірність безвідмовної роботи всіх учасників

$$P\text{учасників} = 1 - (1 - P\text{учасників})^n = 1 - (0.1)^3 = 0.999 \quad (2.5)$$

Визначимо ймовірність безвідмовної роботи учасників і брандмауера

$$P\text{загальне} = P\text{учасників} * P\text{брандмауре} = 0.999 * 0.9 = 0.8991 \quad (2.6)$$

Таблиця 2.1. Вірогідність мережі без резервування

Кількість учасників мережі блокчейн	Кількість елементів резервування брандмауера	Вірогідність безвідмовної роботи мережі за час T
3	0	0,8991
4	0	0,89991
6	0	0,8999991
8	0	0,899999991

Приклад розрахунку вірогідності з ненавантаженим резервуванням:

Визначимо ймовірність безвідмовної роботи всіх учасників

$$P\text{учасників} = 1 - (1 - P\text{учасників})^n = 1 - (0.1)^3 = 0.999 \quad (2.7)$$

Визначимо ймовірність відмови брандмауера при резервуванні на 1.

$$P\text{брандмауре} = 1 - \frac{1}{(k+1)!} * \left(1 - P(T)\right)^{\text{к}+1} = 1 - \frac{1}{2}(0.1)^2 = 0.995 \quad (2.8)$$

де **к** – кількість елементів резервування

Визначимо ймовірність безвідмовної роботи учасників і брандмауера

$$P\text{загальне} = P\text{учасників} * P\text{брандмауре} = 0.999 * 0.995 = 0.99506 \quad (2.9)$$

Таблиця 2.2. Вірогідність мережі з резервуванням на 1

Кількість учасників мережі блокчейн	Кількість елементів резервування брандмауера	Вірогідність відмови мережі за час T
3	1	0.994006
4	1	0,9949005
6	1	0,99499005
8	1	0,99499999

Таблиця 2.3. Вірогідність мережі з резервуванням на 2

Кількість учасників мережі блокчейн	Кількість елементів резервування брандмауера	Вірогідність відмови мережі за час T
3	2	0.9988335
4	2	0,99973335
6	2	0,99983233
8	2	0,9949999

Згідно цих даних можна зробити висновок, що оптимальною буде конфігурація з 3 учасників мережі, та одного резервування для брандмауера.

Завершаючий вигляд системи підключений до SDN мережі буде вигдядати наступним чином:

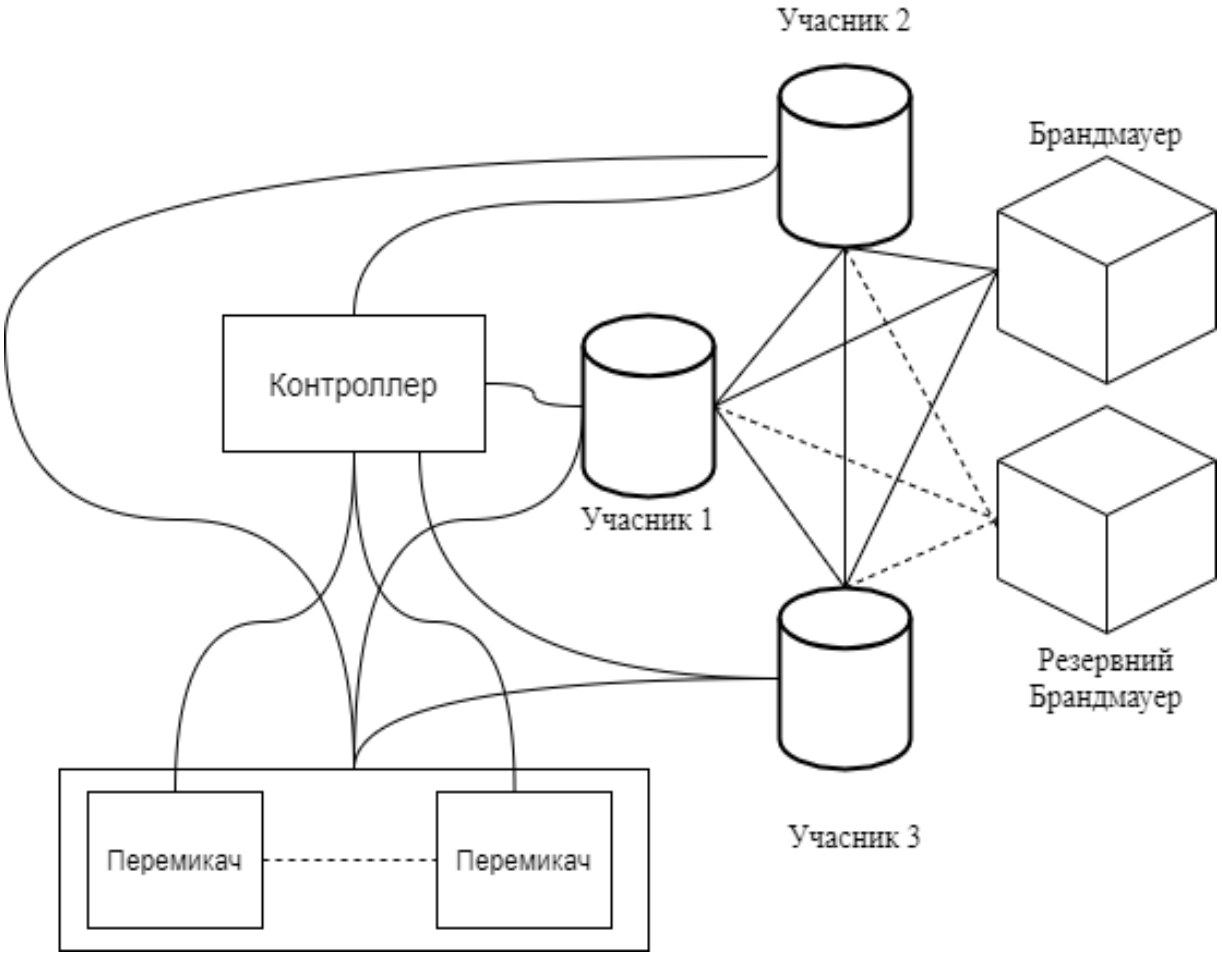


Рис 2.9. Структура всієї схеми системи

2.7 Обробка отриманої лог-інформації

Лог-інформація яку отримуємо з контролера і системи моделяції не є структурованою, її важко порівняти між собою. Тому треба отримати основні компоненти для того щоб їх порівнювати. Для того щоб це зробити, будуть використані регулярні вирази. Вони знаходять шаблоні данні по заданій моделі. Робота цього інструменту базується на кінцевих автоматах. Складається з набору станів, є початковий і завершальний стани, зберігається набір можливих переходів. Швидкодія не дуже підходить для цієї мережі, але немає інших альтернатив, які б дозволили б настільки зручно отримати бажаний результат.

Висновок до розділу 2

Були розглянуті основні компоненти мережі блокчейн, які несуть важливі архітектурні рішення. Вони впливатимуть на безпеку мережі та можливості її вдосконалення в майбутньому. В результаті аналізу вибрані наступні компоненти:

Proof of Authority – як алгоритм консенсусу для блокчейн мережі, завдяки якому визначається вузол в конкретний момент часу який може добавляти блок.

Aura – одна з реалізацій для алгоритму консенсусу Proof of Authority, яка має простий опис роботи.

UDP – протокол для взаємодії вузлів, який наддасть швидкодію розробленій мережі, через відсутність додаткових значень в заголовку.

SHA-3 - хеш алгоритм, як стійкий до атак на колізії алгоритм для хеш-значення блоку.

ECC – асиметричний алгоритм шифрування, який буде використовуватись для цифрового підпису блоку.

JSON - формат даних для передачі між вузлами.

Розробили ключові структури даних для мережі блокчейн. Визначили оптимальну кількість вузлів для того, щоб система була відмовостійкою.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ

3.1 Налаштування моделі всієї мережі

Для того щоб продемонструвати роботоспроможність і розробити програмне забезпечення для справжнього обладнання, буде використовуватись система моделювання. Використаємо SDN контролер стороннього розробника. Загальна кількість компонентів які будуть використовуватись в мережі: одна система моделювання мережі, один контролер, 3 користувачі блокчейн, 1 брандмауер.

3.1.1 Вибір системи моделювання

Для моделювання роботи будемо використовувати симулятор мережі SDN, SDN контролер, середу контейнеризації, розроблену блокчейн мережу.

Для моделювання мережі SDN використовують зазвичай такі симулятори: mininet, maxiNet, DOT, Ns3.

Таблиця 3.1. Опис систем моделювання мереж

Емулятор	Опис
Mininet	Симулює мережу з хостами, перемикачами, підключеннями на одній машині. Зазвичай використовують дане програмне забезпечення у віртуальній машині. Кожен елемент мережі отримує свій mac и IP адрес. Широко використовується тому має велике суспільство до якого можна звернутись. Дає змогу повної конфігурації мережі, дозволяє підключати зовнішні контролери до мережі. Дозволяє налаштувати мережу під будь-яку існуючу версію OpenFlow протоколу.

Таблиця 3.1.

Емулятор	Опис
maxiNet	Симулятор який є розширенням для Mininet, так як дозволяє запускати симуляцію на багатьох фізичних машинах. На кожній машині запускається симуляція яка моделює тільки частину мережі. Дане програмне забезпечення дозволяє моделювати мережі необмежених розмірів.
Ns3	Має схожий функціонал до Mininet, але має більше можливостей для моделювання специфічних сценаріїв. Головною метою цього симулятора є направленість на навчання та досліди.

Серед перечислених симуляторів для цієї роботи використаємо mininet, так як широка аудиторія його використання зможе допомогти при складнощах які можуть виникнути.

Побудуємо довільну мережу. Мережа симуляції буде складатись з одного контролера, 12 хостів, 8 перемикачів.

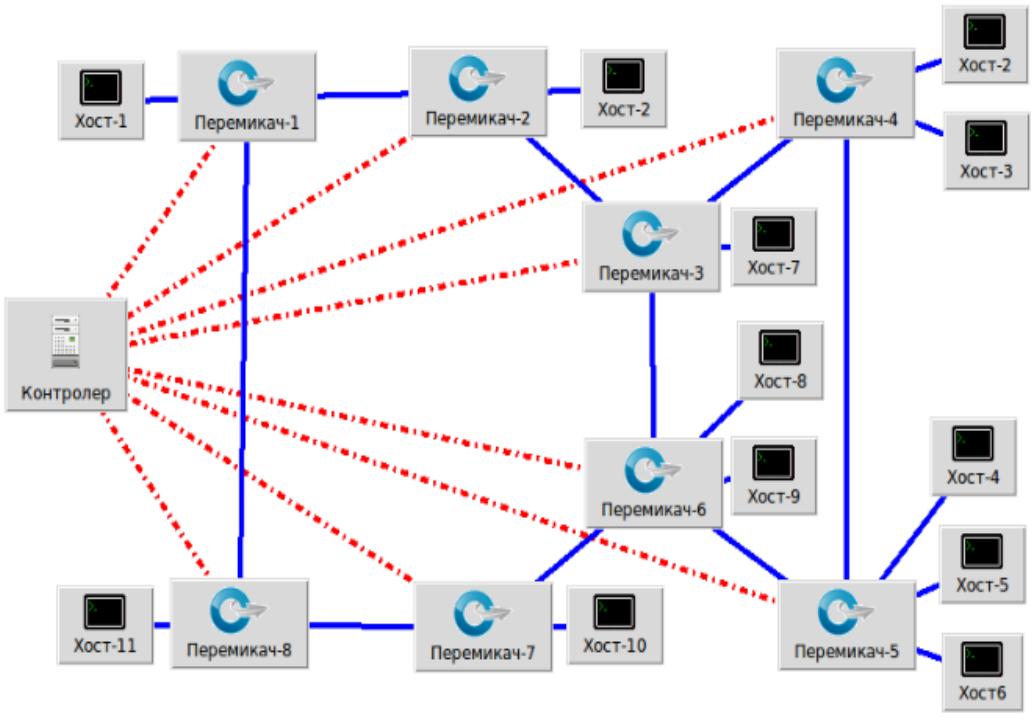


Рис. 3.1. Зображення моделюючої системи в Mininet

3.1.1 Вибір реалізації SDN контролера

Для того щоб система змогла працювати треба налаштувати SDN контролер. Наразі існує багато реалізацій SDN, найбільш поширеними, які все ще підтримуються сторонніми розробниками є наступні контролери: ONOS, Floodlight, OpenDaylight, Beacon, POX.

Таблиця 3.2. Опис контролерів від сторонніх розробників

Контролер	Опис
ONOS	Контролер для мережі SDN, єдиний контролер який підтримує перехід від старих мереж до нових. Дозволяє динамічно додавати та відключати компоненти мережі. Є можливість використовувати кластер контролерів разом з FlowVisor який розподіляє повноваження кожному контролеру. Дозволяє працювати не тільки на OpenFlow протоколі. Надає зручний інтерфейс користувача для отримання загальної статистики мережі. Готових образів контейнеризації немає.
Floodlight	Контролер для мережі SDN, який має підтримує всі версії протоколу OpenFlow. Має можливість для написання власних модулів, для розширення функціональності. Має зручний інтерфейс користувача, дозволяє маніпулювати мережею. Надає зручні контейнеризовані образи.
Beacon	Контролер мережі SDN який розробляться з 2010 року, тому має стабільні версії. Має змогу підтримувати біля 100 перемикачів, 20 – фізичних перемикачів ЦОД. Дозволяє динамічно налаштувати мережі. Надає зручний інтерфейс користувача. Не надає контейнеризовані образи.

Таблиця 3.2

Контролер	Опис
POX	Контролер мережі SDN який підтримує OpenFlow протокол версії 1.0. Розрахований для мереж не великих розмірів. Не надає інтерфейс користувача. Використовує інтерпретатора python, тому швидкодія даного контролера є низькою.
OpenDaylight	SDN – контролер який має модульну систему, може використовуватись до мереж необмежених масштабів. Будує в оперативній пам’яті карту мережі, зо дозволяє виконувати логічні обрахунки. Також можна використати для кластеризації. Надає інтерфейс користувача. Має дуже багата розширень, так як базується на великій кількості інших контролерів. Не надає контейнеризовані образи.

Всі контролери відповідають вимогам роботи з мережею SDN та підтримкою протоколу OpenFlow. Всі контролери окрім POX працюють на віртуальній машині java що дозволяє їм працювати на будь-якій операційній системі. Серед всіх перечислених в цій роботі використаємо Floodlight, так як він надає зручний інтерфейс для отримання даних по мережі. Також даний контролер має образ контейнеру, який дозволить з легкістю запустити його. Для того він почав працювати достатньо однієї команди і перенаправлення портів на відповідні.

3.1.1 Налаштування симуляційного стенду

Використаємо систему контейнеризації Docker на яку не будуть впливати зовнішні процеси. Дана система створить міст, тобто створить підмережу яка буде доступною на цьому же комп’ютері. Також назначить список IP адресів, які будуть доступні в цій мережі. Для того щоб отримувати лог-файли в яких міститься інформація про потокові правила які надав контролер, зробимо

загально доступні дерикторії, до яких будуть мати доступ всі додатки. Після запуску контейнера з контролером в середні, зробимо пере направлення портів. Пере направимо порт 8080 и 6653, так як на порту 8080 запуститься інтерфейс який дозволить відстежувати стан системи, а на порту 6653 запуститься сам контролер. Цю інформації занесемо до симуляційного стенду Mininet.

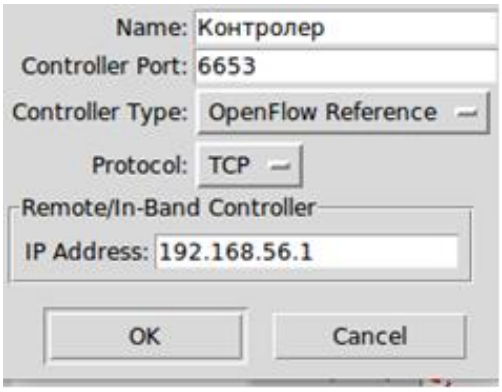


Рис. 3.2. Зображення налаштувань контролера в mininet

Після чого запустимо всі компоненти мережі. Виконаємо перевірку всіх компонентів мережі. Інтерфейс контролера сформує карту мережі, яка має наступний вигляд:

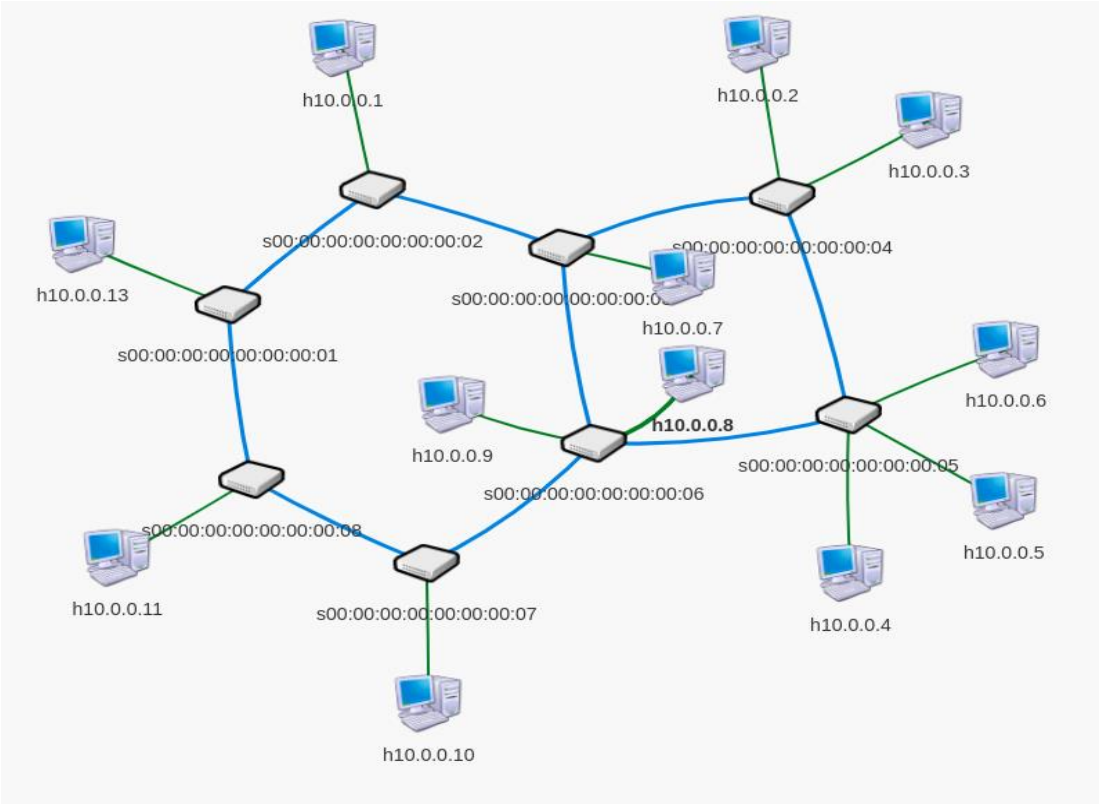


Рисунок 3.3. Зображення моделюючої системи в інтерфейсі контролера

Підчас запуску контролер використовує 256мб оперативної пам'яті. За результатами, контролер знайшов вісім перемикачі, дванадцять хостів, і 18 з’єднань. Отримаємо наступну таблицю IP і mac адресів мережі яку було створено.

Таблиця 3.3. IP та mac адреси компонентів мережі

Назва компоненту	IP-адрес	Mac-адрес
Контролер	192.168.56.1	00:00:00:00:00:00:00:00
Хост-1	10.0.0.13	aa:bb:d2:4b:08:20
Хост-2	10.0.0.1	86:b9:40:62:49:c6
Хост-3	10.0.0.2	5a:e2:af:ec:aa:a2
Хост-4	10.0.0.3	0a:c7:57:d4:98:78
Хост-5	10.0.0.6	ee:e7:d5:08:e2:f1
Хост-6	10.0.0.5	d2:52:b0:24:9a:5b
Хост-7	10.0.0.4	1a:08:9b:d5:03:36
Хост-8	10.0.0.8	ee:ed:8d:6f:0f:b6
Хост-9	10.0.0.9	e6:ae:6e:db:e7:af
Хост-10	10.0.0.10	ae:75:06:e4:43:3e
Хост-11	10.0.0.11	1a:0d:75:9b:fa:ea
Перемикач-1	192.168.56.120:53396	00:00:00:00:00:00:00:01
Перемикач-2	192.168.56.120:53406	00:00:00:00:00:00:00:02
Перемикач-3	192.168.56.120:53404	00:00:00:00:00:00:00:03
Перемикач-4	192.168.56.120:53392	00:00:00:00:00:00:00:04
Перемикач-5	192.168.56.120:53398	00:00:00:00:00:00:00:05
Перемикач-6	192.168.56.120:53402	00:00:00:00:00:00:00:06
Перемикач-7	192.168.56.120:53400	00:00:00:00:00:00:00:06
Перемикач-8	192.168.56.120:53394	00:00:00:00:00:00:00:06

Відправимо тестове повідомлення від Хоста-1 до Хоста-2. В симуляторі mininet отримаємо наступний результат:

Таблиця 3.4. Потокові правила встановленні на тестових перемикачах

Перемикач	Стан таблиці
Перемикач-1	NXST_FLOW reply (xid=0x4) cookie=0x20000129000000, duration=4.861s, table=0, n_packets=3, n_bytes=294, idle_timeout=5, idle_age=1, priority=1, ip,in_port=3, dl_src=86:b9:40:62:49:c6, dl_dst=aa:bb:d2:4b:08:20, nw_src=10.0.0.1, nw_dst=10.0.0.13 actions=output:1 cookie=0x20000128000000, duration=4.863s, table=0, n_packets=3, n_bytes=294, idle_timeout=5, idle_age=1, priority=1, ip, in_port=1, dl_src=aa:bb:d2:4b:08:20, dl_dst=86:b9:40:62:49:c6, nw_src=10.0.0.13, nw_dst=10.0.0.1 actions=output:3
Перемикач-2	NXST_FLOW reply (xid=0x4): cookie=0x20000129000000, duration=4.872s, table=0, n_packets=3, n_bytes=294, idle_timeout=5, idle_age=1, priority=1, ip, in_port=1, dl_src=86:b9:40:62:49:c6, dl_dst=aa:bb:d2:4b:08:20, nw_src=10.0.0.1,nw_dst=10.0.0.13 actions=output:2 cookie=0x20000128000000, duration=4.874s, table=0, n_packets=4, n_bytes=392, idle_timeout=5, idle_age=1,priority=1, ip, in_port=2, dl_src=aa:bb:d2:4b:08:20, dl_dst=86:b9:40:62:49:c6, nw_src=10.0.0.13, nw_dst=10.0.0.1 actions=output:1

Таблиця 3.5. Визначення параметрів потокових таблиць

Параметр	Призначення
cookie	64-бітне унікальне значення, яке асоціюється з потоком в таблиці потоків. Значення за замовчуванням дорівнює нулю.
duration	Визначає час протягом якого запис залишається в таблиці. Є можливість встановити час з точність до мілісекунд.
n_packets	Кількість пакетів які збіглись з записом в таблиці потоків.
n_bytes	Загальна кількість байт з отриманих пакетів які збігаються з записом в таблиці потоків.
idle_timeout	Встановлює час протягом якого буде існувати потоку в таблиці. Якщо значення 0, то запис не має граничного терміну.
idle_age	Числове значення секунд які пройшли без жодного пакету
priority	Числове значення між 0 і 65535, в залежності від значення буде залежить порядок перевірки.
in_port	Номер порту за яким відбувається обробка дій.
dl_src	MAC – адреса відправника пакету
dl_dst	MAC – адреса отримувача пакету
nw_src	IP – адрес відправника пакету
nw_dst	IP – адрес того хто отримує пакет
actions	Список дій які застосовуються до пакету коли потокові правила співпадають. Якщо дія не визначення, то пакет відсилається до контролера або відкидається, в залежності від налаштування мережі.

Згідно до таблиці правил яку ми отримали, можна зробити висновок, що Перемикач-1 налаштований на передачу пакету від хоста 10.0.0.13 до хоста 10.0.0.13, також цю дію продубльовано в зворотному напрямку. Для

Перемикач-2 бачимо відповідно передачу пакету від хоста 10.0.0.1 до хоста 10.0.0.13. Дана інформація підтверджує коректність роботи мережі побудованої на симуляторі mininet.

Проаналізуємо лог-файли отримані від контролера.

Таблиця 3.6. Лог-інформація отримана від контролера

Контролер	Лог-інформація
FloodLight контролер	Pushing Route flowmod routeIndx=3 sw=OFSwitch DPID[00:00:00:00:00:00:00:01] inPort=3 outPort=1 [nioEventLoopGroup-3-10] Not dampening new msg OFFlowAddVer10(xid=6191, match=OFMatchV1Ver10(in_port=3, eth_src=86:b9:40:62:49:c6, eth_dst=aa:bb:d2:4b:08:20, eth_type=0x800, ipv4_src=10.0.0.1, ipv4_dst=10.0.0.13), cookie=0x0020000137000000, idleTimeout=5, hardTimeout=0, priority=1, bufferId=4294967295, outPort=1, flags=[], actions=[OFActionOutputVer10(port=1, maxLen=2147483647)])

З даної інформації видно, що на перемикач 00:00:00:00:00:00:00:01 було встановлено потокове правило на передачу пакету від хоста 10.0.0.1 до хоста 10.0.0.13.

Розроблена система використовує різні мережеві інтерфейси, тим самим відображає більш правдоподібну систему, відповідно до реальних. Взаємодія інтерфейсів розробленої мережі зображена на (рис. 3.4).

Рис. 3.4. Зображення взаємодії мережевих інтерфейсів

Висновок до розділу 3

Визначили систему емуляції мережі SDN, побудова тестової мережі буде відбуватись в mininet симуляторі. Визначили контролер для тестової мережі, з якого і буде братись лог-інформація. Запустили контролер FloodLight в контейнеризованому просторі. Визначили лог-записи які є важливими для системи. Побудували тестову схему. Визначили відмовостійкість системи. Система відповідає реальній мережі до якої є можливість підключити блокчейн і проводити моніторинг для забезпечення безпеки.

РОЗДІЛ 4
ТЕСТУВАННЯ СИСТЕМИ

4.1 Відображення результату працездатності розробленої системи

В ході тестування були виявленні недоліки при скануванні лог-інформації від контролера. Значні витрати ресурсів витрачаються на зчитування цих даних, тому кількість інформації яка зчитується була обмежена 1500 лініями. Через це виникла проблема. Так як контролер може не видавати правила нові, але вони залишаються на перемикачах. Через це виникла необхідність зберігати в оперативній пам’яті команди контролера, а видаляти їх звідти тільки коли контролер очистить потокові правила на перемикачі. Так як в ході тестування виявилось, що контролер очищує потокові правила, якщо шляхи маршрутизації не використовуються.

Розроблений брандмауер будує топологію мережі блокчейн, на основі інформації одного вузла. Приклад побудованої системи на рис. 4.1.

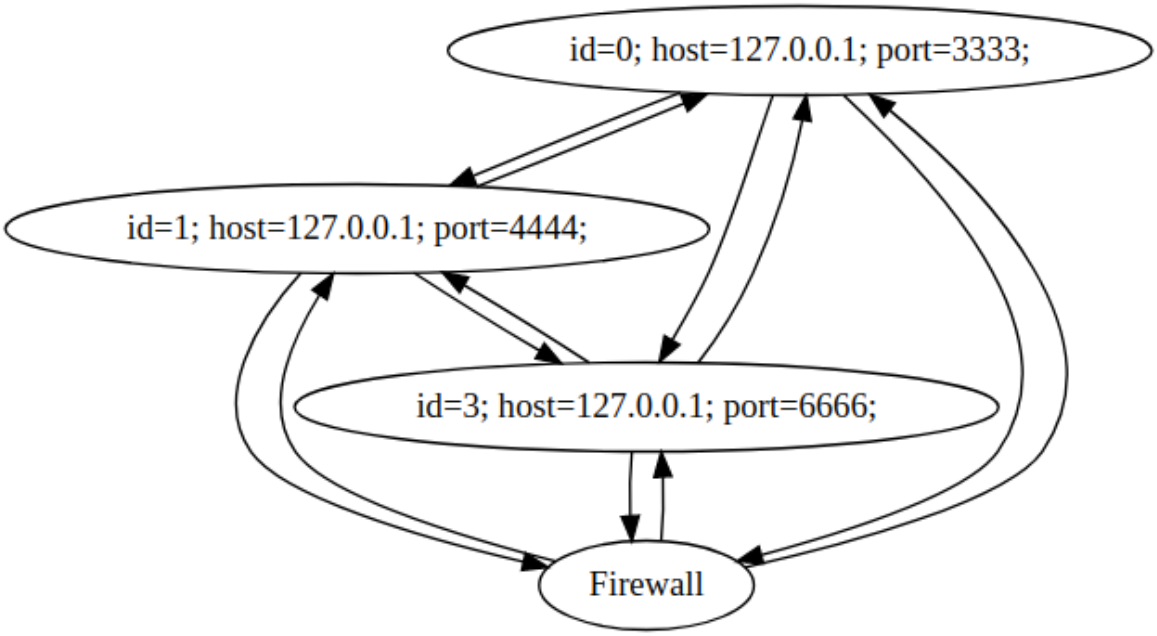


Рис. 4.1 Побудована система безпеки на основі інформації на одному вузлі

Результати транзакцій які записані в блоки, відображені у вигляді таблиці. Поля позначені як червоні відображають втручання в роботу системи, яку потрібно перевірити адміністратору мережі. Зелений колір відображає що всі операції в мережі SDN підтвердженні. Результати наведені на (рис. 4.2 та 4.3).

Хеш	Час створення	Хеш минулого блоку	Транзакції
4868232111...	1589314419961		
4868232461...	1589314430003	4868232111...	
4869233013...	1589314460000	4868232461...	
4869233023...	1589314490002	4869233013...	
4868232582...	1589314520000	4869233023...	{"switchId":2,"mininetLog": {"src":"10.0.0.1","dst":"10.0.0.2","actions":"output:3"},"controllerLog": {"src":"10.0.0.1","dst":"10.0.0.2","actions":"output:3"},"result":true} {"switchId":2,"mininetLog": {"src":"10.0.0.2","dst":"10.0.0.1","actions":"output:1"},"controllerLog": {"src":"10.0.0.2","dst":"10.0.0.1","actions":"output:1"},"result":true} {"switchId":1,"mininetLog": {"src":"10.0.0.1","dst":"10.0.0.2","actions":"output:1"},"controllerLog": {"src":"10.0.0.1","dst":"10.0.0.2","actions":"output:1"},"result":true} {"switchId":1,"mininetLog": {"src":"10.0.0.2","dst":"10.0.0.1","actions":"output:3"},"controllerLog": {"src":"10.0.0.2","dst":"10.0.0.1","actions":"output:3"},"result":true}

Рис. 4.2 таблиця блоків які відображаюсь задовільні транзакції мережі

4868232657...	1589314610001	4868232541...	{"switchId":3,"mininetLog": {"src":"10.0.0.8","dst":"10.0.0.1","actions":"output:1"},"controllerLog": {},"result":false} {"switchId":3,"mininetLog": {"src":"10.0.0.1","dst":"10.0.0.8","actions":"output:4"},"controllerLog": {},"result":false} {"switchId":2,"mininetLog": {"src":"10.0.0.8","dst":"10.0.0.1","actions":"output:1"},"controllerLog": {},"result":false} {"switchId":2,"mininetLog": {"src":"10.0.0.1","dst":"10.0.0.8","actions":"output:2"},"controllerLog": {},"result":false} {"switchId":1,"mininetLog": {"src":"10.0.0.8","dst":"10.0.0.1","actions":"output:3"},"controllerLog": {},"result":false} {"switchId":1,"mininetLog": {"src":"10.0.0.1","dst":"10.0.0.8","actions":"output:1"},"controllerLog": {},"result":false}

Рис. 4.3 таблиця блоків які відображаюсь втручання в мережу

Система надає велику швидкодія, більшість втручання були помічені. Помилки виникали при великій завантаженості мережі, так як в цей момент велика кількість лог-інформації записується контролером. Для того щоб виправити цю проблему не достатньо збільшити кількість вузлів блокчейн які аналізують лог-інформацію, тому що виникає блокування на читання при великій кількості учасників. Для вирішення цього оптимально буде використати кластер SDN контролерів, які будуть відповідальні за частину мережі. В цьому випадку буде достатньо по 3 вузли які контролюють один контролер. Також можна використати 3 вузли блокчейна під кластерну систему SDN, в цьому випадку блокчейн збереже свою швидкодію, так як він побудований асинхронному вводі і виводі, що забезпечує не блокувати потік на момент коли данні ще не готові, тим самим вузол може зчитувати в цей момент лог-інформацію з іншого вузла.

В ході тестування оптимальний час для генерації блоку є 10 секунд. Система дозволяє масштабувати її, можна встановити будь-який інтервал, чим буде він меншим тим буде більше точність мережі, але збільшиться затрати по пам’яті та ресурсам комп’ютера.

4.2 Пропозиції по вдосконаленню системи

Побудована система відповідає всім потребам безпеки, та піддається модифікації. Будь-яка зміна не вплине на роботу мережі блокчейн. В ході тестування були виявленні недоліки побудованої системи, які не є критичними для її функціонування. Але для майбутнього вдосконалення вони будуть актуальними.

Обмежити довжину блокчейну, більшість інформації яка зберігається в блокчейні з часом стає не потрібною, вона починає займати тільки простір в пам’яті. Для цієї системи є важливим інформація тільки за останній час, а не вся історія. Тому для оптимізації можна зробити обмеження на довжину ланцюгу, при досягненні ліміту, буде назначити новий генеруючий блок

мережі, який буде взятий з середини ланцюгу, через голосування можна розробити відповідний алгоритм.

Також можна не просто видаляти данні, а зберігати їх в довгострокову пам’ять. Це зменшить витрати на оперативну пам’ять, при потребі дозволить продивитись всі транзакції, або отримати статистику за певний період по мережі.

Розробити функціонал для блокування IP адресів які можливо внесли данні шкідливі правила в перемикач.

Також обмежити доступ до брандмауера через авторизації або автентифікацію.

Висновок до розділу 4

Провели тестування розробленої системи, впевнились в її можливості працювати. Отримали схему мережі на основі інформації з одного вузла. Отримали список блок з транзакціями. Вся інформація відображена в зручному вигляді для роботи з нею. Визначили основні недоліки та запропонували покращення мережі у майбутньому.

ВИСНОВКИ

Робота присвячена розробці підходу по забезпеченню безпеки у мобільних мережах. Було проведено огляд та аналіз підходів по вирішенню проблем традиційних мереж. Виявлені основні загрози на кожен компонент мережі SDN. Проаналізовані існуючі рішення та концепції впровадження безпеки в SDN мережі.

Вибраний блокчейн підхід як один з ефективних методів по забезпеченню безпеки. Проаналізована структура блокчейн мережі, визначення архітектура та засоби взаємодії блокчейн мережі для досягнення найкращої швидкодії. Визначена оптимальна кількість вузлів для роботи такої системи. Розглянуті засоби інтеграції блокчейн та SDN мережі. Визначні хеш-алгоритми для забезпечення консистентності та цілісності даних та алгоритми цифрового підпису для підтвердження створеного блоку учасниками мережі. Проаналізовані алгоритми консенсусу для мережі блокчейн.

Визначений компонент симуляції мережі та контролер який буде керувати нею. Розроблений механізм зчитування лог-інформації з перемикачів та контролера. Розроблений інструмент обробки отриманої інформації для представлення результатів у зручному вигляді. Розроблений брандмауер сервер для зручної взаємодії адміністратора мережі з інструментом забезпечення безпеки.

Проведе тестування механізму, визначені основні недоліки системи та запропоновані шляхи по вдосконаленню мережі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Slavica Tomovic, Milica Pejanovic-Djurisic, Igor Radusinovic, SDN Based Mobile Networks: Concepts and Benefits

2. Е.В. Дуравкин, Е.Б. Ткачева, Иссам Саад. АРХИТЕКТУРА SDN. АНАЛИЗ ОСНОВНЫХ ПРОБЛЕМ НА ПУТИ РАЗВИТИЯ

3. Виртуализация сетей в Linux. [Електронний ресурс]. – Режим доступу: URL: https://www.ibm.com/developerworks/ru/library/l-virtual-networking (Дата звернення 11.05.2020).

4. The difference between SDN and NFV. [Електронний ресурс]. – Режим доступу: URL: https://codilime.com/the-difference-between-sdn-and-nfv-a-simple-guide/ (Дата звернення 11.05.2020).

5. SDN и NFV: как это работает на сети оператора связи. [Електронний ресурс]. – Режим доступу: URL: https://shalaginov.com/2015/12/27/sdn-и-nfv-как-это-работает-на-сети-оператора/ (Дата звернення 11.05.2020)

6. Ijaz Ahmad, Suneth Namal, Mika Ylianttila, Senior Member, IEEE, and Andrei Gurtov, Senior Member, IEEE. Security in Software Defined Networks: A Survey / IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 4, FOURTH QUARTER 2015.

7. Levent Ertaul, Krishnakumar Venkatachalam, Security of Software Defined Networks / IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 4, FOURTH QUARTER 2015.

8. Ryan Beckett, X. Kelvin Zou, Shuyuan Zhang, Sharad Malik, Jennifer Rexford, David Walker / An Assertion Language for Debugging SDN Applications.

9. Phillip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, Guofei Gu / A Security Enforcement Kernel for OpenFlow Networks

10. Ehab AI-Shaer, Saeed AI-Haj/ FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures.

11. Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, P. Brighten Godfrey/ VeriFlow: Verifying Network-Wide Invariants in Real Time.

12. Abdelouahid Derhab, Mohamed Guerroumi, Abdu Gumaei, Leandros Maglaras, Mohamed Amine Ferrag, Mithun Mukherjee, Farrukh Aslam Khan / Blockchain and Random Subspace Learning-Based IDS for SDN-Enabled Industrial IoT Security.

13. Joseph J. Bambara, Paul R. Allen. Blockchain. A Practical Guide to Developing Business, LAW, and Technology Solutions. С. 2-7.

14. Imar Bashir, Mastering Blockchain Second Edition. Випуск березень 2018. С. 73 – 76

15. Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda, Beginning Blockchain. Випуск 2018. С. 35

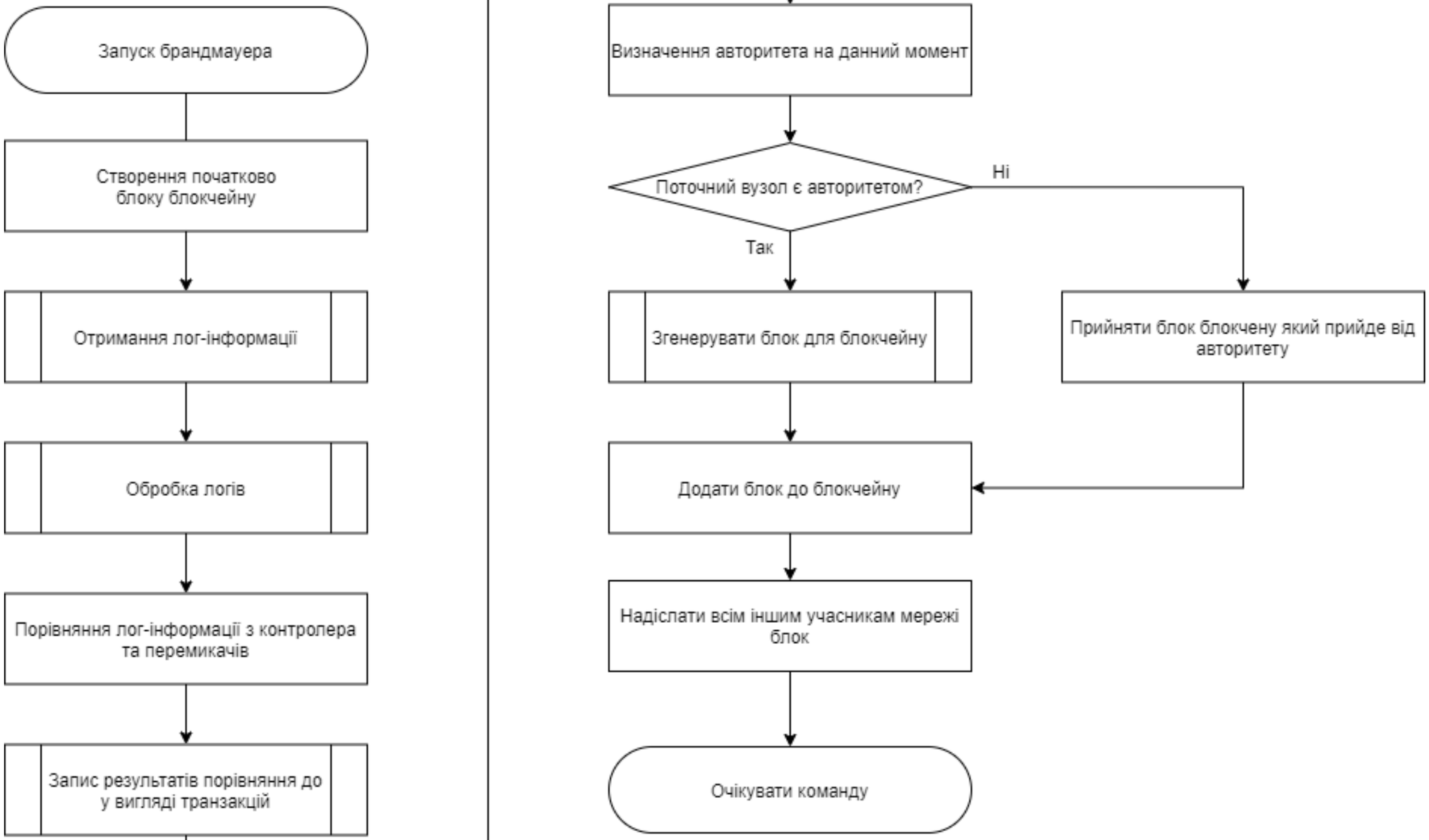
16. Joseph J. Bambara, Paul R. Allen. Blockchain. A Practical Guide to Developing Business, LAW, and Technology Solutions. С. 13-15.

17. Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombard, Andrea Margheri, Vladimiro Sassone, PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain.

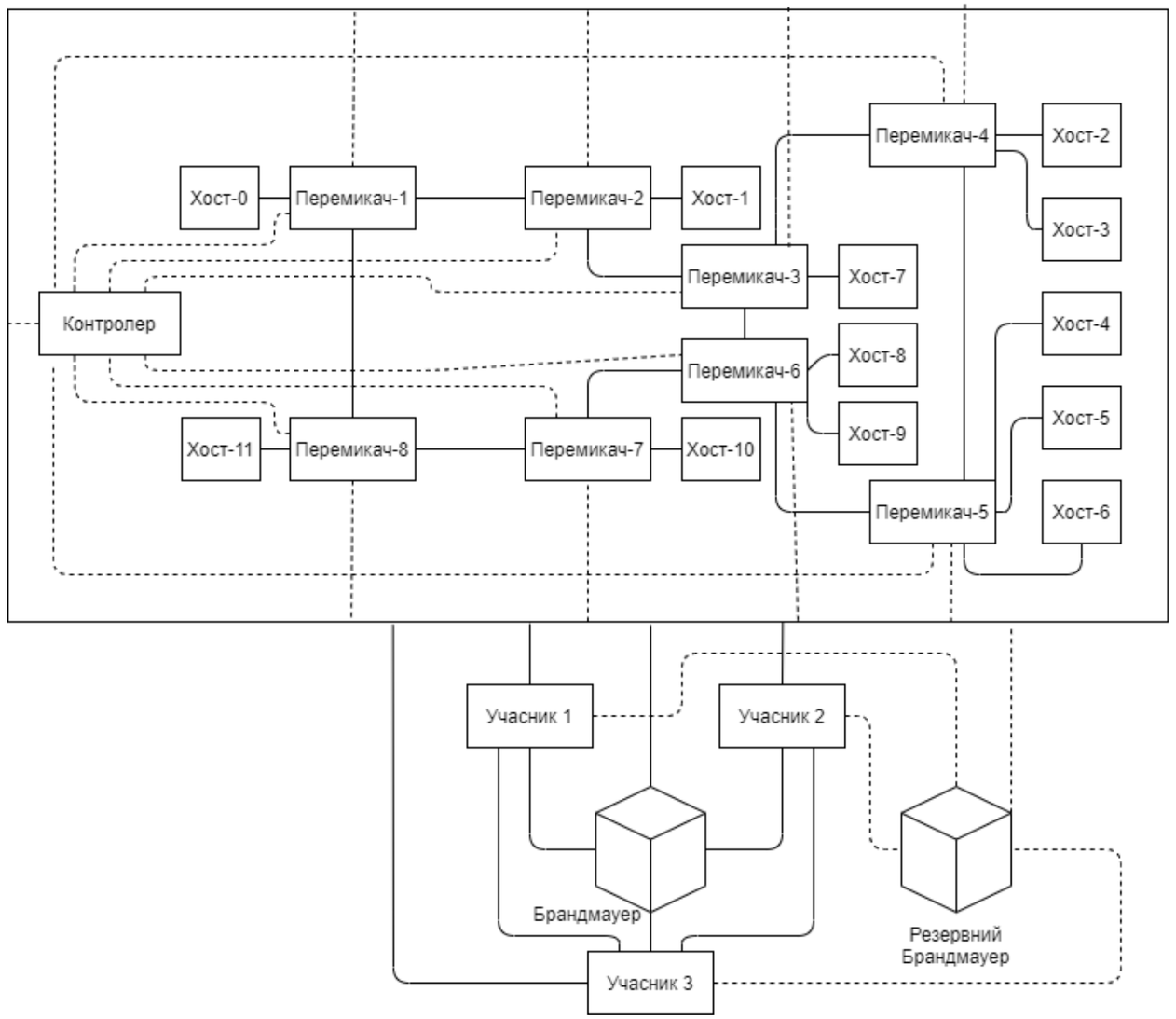
18. TCP vs UDP/. [Електронний ресурс]. – Режим доступу: URL: http://web.cecs.pdx.edu/~jsnow/wireless_performance/tcp_udp.html (Дата звернення 11.05.2020)

19. Bart Preneel, Understanding Cryptography. Випуск вересень 2009.

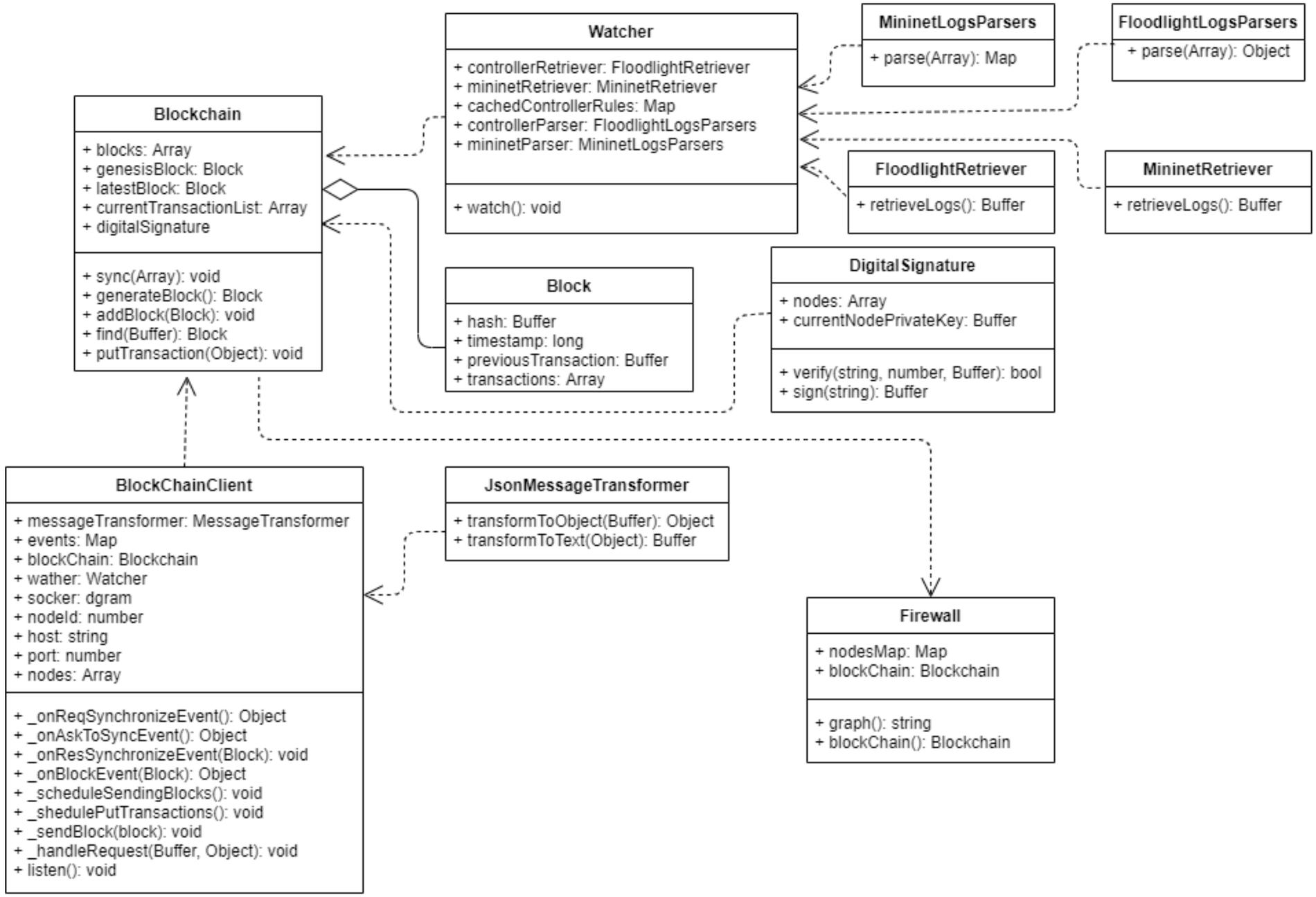
20. Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda, Beginning Blockchain. Випуск 2018. С.117-120



					ДП 045430. 03.000 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата	Схема програмного забезпечення Схема принципова	Літера / Маса / Масштаб
Розроб.		Воробйов А.О.				
Перевір.		Кулаков Ю.О.				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоменко В.П.				НТУУ КПІ ФІОТ Група ІО-61
Затв.						



					ДП 045430. 04.000 Д2	
					Схема мережі Схема структурна	Літера: — Маса: — Масштаб: —
Зм.	Арк.	№ докум.	Підпис	Дата		Аркуш 1 / Аркушів 1
Розроб.		Воробйов А.О.				НТУУ КПІ ФІОТ Група ІО-61
Перевір.		Кулаков Ю.О.				
Т. контр.						
Н. контр.		Сімоменко В.П.				
Затв.						



					ДП 045430. 05.000 ДЗ
Зм.	Арк.	№ докум.	Підпис	Дата	Схема блокчейну з брандмауером Схема функціональна
Розроб.		Воробйов А.О.			
Перевір.		Кулаков Ю.О.			
Т. контр.					
Н. контр.		Сімоменко В.П.			
Затв.					

Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

НТУУ КПІ ФІОТ
Група ІО-61

ДОДАТОК А

Лістинг програми

					ДП 045430 06.000.00 ДА			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Воробйов А.О.			Лістинг програми	Літ.	Аркуш	Аркушів
Перевірив		Кулаков Ю.О.					1	16
Реценз.						НТУУ КПІ, ФІОТ, ІО-		
Н. Контр.		Сімоненко В. П.						
Затвердив								

```
const Block = require('./Block');
const {normalizeBlock} = require("../util");
const {calculateHash} = require("../util");

module.exports = function crateBlockChain(digitalSignature) {
  return new RChainBuilder(digitalSignature);
};

class RChain {
  constructor(genesisBlock, latestBlock, blocks, digitalSignature) {
    this.blocks = blocks;
    this.genesisBlock = genesisBlock;
    this.latestBlock = latestBlock;
    this.currentTransactionList = [];
    this.digitalSignature = digitalSignature;
  }

  sync = async (blocks) => {
    if (blocks.length === 0) return false;
    if (blocks[0].hash.toString() !== this.genesisBlock.hash.toString()) {
      throw new Error("Cannot sync chains")
    }
    const retrievedChainIsLonger = blocks.length > this.blocks.length;
    if (retrievedChainIsLonger) {
      const isValid = await this._validateBlocks(blocks);
      if (!isValid) {
        throw new Error("Blocks are not valid")
      }

      const blocksWithTransactionsWhichAddToNewBlock = [];
      for (let i = 0; i < blocks.length; i++) {
        if (this.blocks[i] && blocks[i].hash.toString() !==
this.blocks[i].hash.toString()) {
          blocksWithTransactionsWhichAddToNewBlock.push(this.blocks[i])
        }
        this.blocks[i] = blocks[i];
      }
      blocksWithTransactionsWhichAddToNewBlock.forEach(block => {
        this.currentTransactionList.push(...block.transactions);
      })
      return true;
    } else {
      return false;
    }
```

```
  }

  _validateBlocks = async (blocks) => {
    return Promise.all(
      blocks.map((block) => {
        const {transactions, timestamp, hash, previousHash} = block;
        const calculatedHash = calculateHash(timestamp, transactions,
previousHash);
        return this.digitalSignature.verify(calculatedHash, timestamp, hash);
      })
    )
      .then(() => true)
      .catch(() => false);
  }

  generateBlock = async () => {
    const generatedBlock = await Block.initBlock(
      this.currentTransactionList,
      this.latestBlock.hash,
      this.digitalSignature
    );
    this.currentTransactionList = [];
    this.blocks.push(generatedBlock);
    return this.latestBlock = generatedBlock;
  }

  addBlock = async (block) => {
    const {transactions, timestamp, hash, previousHash} = block;
    const calculatedHash = calculateHash(timestamp, transactions, previousHash);
    const isValid = await this.digitalSignature.verify(calculatedHash, timestamp,
hash);
    if (isValid) {
      this.blocks.push(block);
      return;
    }
    throw new Error("Block is not valid");
  }

  find = (expectedHash) => {
    for (const block of this.blocks) {
      if (block.hash.toString() === expectedHash) {
        return block;
      }
    }
```

```
    return null;
  }

  putTransaction = (transaction) => {
    this.currentTransactionList.push(transaction);
  }
}

class RChainBuilder {
  constructor(digitalSignature) {
    this.digitalSignature = digitalSignature;
  }

  generateGenesisBlock = () => {
    const digitalSignatureForBuild = this.digitalSignature;
    return {
      build: async () => {
        const genesisBlock = await Block.initBlock([], null,
digitalSignatureForBuild);
        return new RChain(genesisBlock, genesisBlock, [genesisBlock],
digitalSignatureForBuild);
      }
    }
  };

  sync = (blockChainSyncer) => {
    const digitalSignatureForBuild = this.digitalSignature;
    return {
      build: async () => {
        const blocks = await blockChainSyncer();
        if (blocks.length === 0) {
          throw new Error("Cannot sync empty chain");
        }
        const normalizedBlocks = blocks.map(block => normalizeBlock(block));
        const genesis = normalizedBlocks[0];
        const latest = normalizedBlocks[normalizedBlocks.length - 1];
        return new RChain(genesis, latest, normalizedBlocks,
digitalSignatureForBuild);
      }
    }
  }
}
```

```
const {calculateHash} = require("../util");

module.exports = class Block {
   constructor(timestamp, transactions, previousHash, hash) {
      this.hash = hash;
      this.timestamp = timestamp;
      this.previousHash = previousHash;
      this.transactions = transactions;
   }

   static async initBlock(transactionList, previousHash, digitalSignature) {
      const timestamp = Date.now();
      const calculatedHash = calculateHash(timestamp, transactionList,
previousHash);
      const signedHash = await digitalSignature.sign(calculatedHash);
      return new Block(timestamp, transactionList, previousHash, signedHash);
   }
};

const schedule = require('node-schedule');
const dgram = require('dgram');
const {normalizeBlock} = require("../util");
const {calculateHash} = require("../util");
const {logger, CustomMap, getAllMethods} = require('../util');

const TIME_TO_BLOCK = process.env.TIME_TO_BLOCK || 10000;
const SECONDS_TIME_TO_BLOCK = (TIME_TO_BLOCK) / 1000;
const FALLBACK_EVENT = (msg, event) => {
  return () => {
    logger.warn(`Event is not found ${event}, with msg: ${msg.toString()}`);
  }
}
const EVENTS = {
  reqSync: "ReqSynchronize",
  resSync: "ResSynchronize",
  block: "Block",
  askToSync: "AskToSync"
};

class BlockChainClient {
  constructor({nodeId, socket, host, port, blockChain, messageTransformer,
watcher}) {
    this.messageTransformer = messageTransformer;
    this.events = new CustomMap();
```

```
  this.blockChain = blockChain;
  this.watcher = watcher;
  this.socket = socket;
  this.nodeId = nodeId;
  this.host = host;
  this.port = port;
  this.nodes = [];
}

withNode = (node) => {
  this.nodes.push(node);
  return this;
}

withNodes = (nodes) => {
  this.nodes.push(...nodes);
  return this;
}

_onReqSynchronizeEvent = (body) => {
  return {event: EVENTS.resSync, blocks: this.blockChain.blocks}
}

_onAskToSyncEvent = () => {
  return {event: EVENTS.reqSync}
}

_onResSynchronizeEvent = async (body) => {
  const {request} = body;
  const blocks = request.blocks.map(block => normalizeBlock(block));
  const isSynced = await this.blockChain.sync(blocks);
  logger.info(`Chain synced ${isSynced}`);
}

_onBlockEvent = async (body) => {
  logger.info("Add new block")
     const {request} = body;
  const block = normalizeBlock(request.block);
  const {previousHash} = block;

  const {hash : latestHash} = this.blockChain.latestBlock;
  if (previousHash.toString() === latestHash.toString()) {
    await this.blockChain.addBlock(block);
    await this._sendBlock(block);
```

```
      return;
    }
    const foundPreviousBlock = this.blockChain.find(previousHash.toString());
    const acceptedBlockWithShortChain = !!foundPreviousBlock;
    return acceptedBlockWithShortChain ? {event: EVENTS.askToSync} : {event:
EVENTS.reqSync};
  }

  _scheduleSendingBlocks = () => {
    schedule.scheduleJob(`*/${SECONDS_TIME_TO_BLOCK} * * * * *`, async
(fireDate) => {
      const currentUserAllowedAddBlock = Math.floor(fireDate /
TIME_TO_BLOCK) % this.nodes.length;
      logger.info(`Schedule sending with date ${fireDate.getTime()}`)
      logger.info(`Current user id is ${currentUserAllowedAddBlock}`)
      if (this.nodeId === currentUserAllowedAddBlock) {
        const newBlock = await this.blockChain.generateBlock();
        await this._sendBlock(newBlock);
      }
    });
  }

  _schedulePutTransactions = () => {
    this.watcher.on("transactions", (transactions) => {
      transactions.forEach(transaction =>
this.blockChain.putTransaction(transaction));
    });
  }

  _sendBlock = async (block) => {
    const transformedBlock = await
this.messageTransformer.transformToText({event: EVENTS.block, block});
    this.nodes.forEach(node => {
      this.socket.send(transformedBlock, node.port, node.host);
    });
  }

  _handleRequest = async (msg, rinfo) => {
    logger.info(`Request from {host: ${rinfo.address}, port: ${rinfo.port}}`)
    const node = this.nodes.find(
      value => value.host === rinfo.address && value.port === rinfo.port
    );
    if (!node) {
      logger.error('Request from node that is not in allowed list')
```

```
      return;
    }
    try {
      const resultRequest = await this.messageTransformer.transformToObject(msg);
      const event = this.events.getOrDefault(resultRequest.event,
FALLBACK_EVENT(msg, resultRequest.event));
      logger.info(`Event is ${resultRequest.event}`)
      const resp = await event({request: resultRequest, personInfo: rinfo});
      if (resp) {
        const resultResponse = await
this.messageTransformer.transformToText(resp);
        return this.socket.send(resultResponse, rinfo.port, rinfo.address)
      }
    } catch (e) {
      logger.error(e)
    }
  }

  listen = async () => {
    if (!this.nodes.length) throw new Error("Nodes are not specified");
    await this._registerEvents();
    await this._registerScheduling();
    this.socket.on('message', this._handleRequest);
    this.socket.bind(this.port, () => {
      logger.info("Client started listening")
      this.watcher.watch();
    });
  };

  _registerEvents = async () => {
    const allMethods = getAllMethods(this);
    allMethods.forEach(method => {
      const matchObj = new RegExp('_on(.+?)Event', 'g').exec(method);
      if (matchObj) {
        this.events.set(matchObj[1], this[method]);
      }
    });
  }

  _registerScheduling = async () => {
    const allMethods = getAllMethods(this);
    allMethods.forEach(method => {
      const matchObj = new RegExp('_schedule(.*?)', 'g').exec(method);
      if (matchObj) {
```

```
        this[method]();
      }
    });
  }
}

module.exports = {
  createBlockChainClient: async ({nodeId, host, port, blockChain,
messageTransformer, watcher}) => {
    const socket = dgram.createSocket('udp4');
    return new BlockChainClient({nodeId, socket, host, port, blockChain,
messageTransformer, watcher});
  },
  initBlockChainSyncer: async (messageTransformer, nodes, host, port) => {
    return async () => {
      const socket = dgram.createSocket('udp4');
      socket.bind(port, host, async () => {
        const result = await messageTransformer.transformToText({event:
EVENTS.reqSync});
        nodes.forEach(node => socket.send(result, node.port, node.host));
      });

      const handleSyncResponse = (resolve, reject) => {
        let isComplete = false;
        setTimeout(() => {
          if (!isComplete) {
            isComplete = true
            reject(new Error("Cannot sync data"))
          }
        }, 5000);
        return async (msg, rinfo) => {
          const node = nodes.find(
            value => value.host === rinfo.address && value.port === rinfo.port
          );
          if (!node) return;
          const result = await messageTransformer.transformToObject(msg);
          if (result.event === EVENTS.resSync) {
            isComplete = true;
            resolve(result.blocks);
            socket.close();
          }
        }
      }
      return new Promise((resolve, reject) => {
```

```
        socket.on('message', handleSyncResponse(resolve, reject))
      });
    }
  }
};

module.exports = {
   JsonMessageTransformer: class JsonMessageTransformer {
      async transformToObject(data) {
         return JSON.parse(data.toString());
      }

      async transformToText(data) {
         return JSON.stringify(data)
      }
   }
};

const express = require('express')
const app = express()
const cors = require('cors');

module.exports = (nodesMap, blockChain) => {
  app.use(cors())

  const staticMiddleware = express.static('public');
  app.use('/static', (req, res, next) => {
    staticMiddleware(req, res, next);
  });

  app.get('/graph', (req, res) => {
    return res.status(200)
      .json({graph: buildGraph(nodesMap)})
  });

  app.get('/blockChain', (req, res) => {
    return res.status(200)
      .json(blockChain.blocks);
  });

  app.listen(8081);
}

const allowParams = ['host', 'port', 'id'];
```

```
function objToString(obj) {
  let str = '';
  for (const p in obj) {
    if (obj.hasOwnProperty(p) && allowParams.includes(p)) {
      str += p + '=' + obj[p] + '; ';
    }
  }
  return str;
}

function buildGraph(nodesMap) {
  const nodesKey = Object.keys(nodesMap);
  const nodesConnections = [];
  for (let key of nodesKey) {
    const node = nodesMap[key];
    nodesConnections.push(node.isFirewall ?
      `Вузол${node.id}[label="Firewall"]` :
      `Вузол${node.id}[label="${objToString(node)}"]`
    );
    const joinNodes = nodesKey
      .filter(joinKey => joinKey !== key)
      .map(key => `Вузол${nodesMap[key].id}`)
      .join(', ');
    nodesConnections.push(`Вузол${nodesMap[key].id} -> {${joinNodes}}`)
  }
  return `digraph D {
      ${nodesConnections.join('\n')}
    }`;
}

const floodLightRegexp = /TRACE.+?OFConnection\s\[00:00:00:00:00:00:00:([0-
9]+?)\(.*?send\sOFFlowAddVer.*?ipv4_src=(.*?),\sipv4_dst=(.*?)\).*?,\sactions=\
[OFAction(\w+)Ver.*?\(.*?port=([0-9]*),/g;
const swithesRegexp =
/\s+cookie.*?nw_src=(.+?),nw_dst=(.+?)\sactions=(.+?)(\n|$)/g;
const floodLightClearLogsRegexp = /Clearing flow tables of
00:00:00:00:00:00:00:(.+) on upcoming transition to/;

module.exports = {
  FloodLightLogsParsers: {
    parse: async (logs) => {
      let result;
      const newRules = new Map();
```

```
    while (result = floodLightRegexp.exec(logs)) {
      const logObject = {src: result[2], dst: result[3], actions:
`${result[4].toLowerCase()}:${result[5]}`};
      const switchId = parseInt(result[1], 10);
      const switchLogs = newRules.get(switchId) || [];
      switchLogs.push(logObject);
      newRules.set(switchId, switchLogs)
    }
    const clearRules = [];
    while (result = floodLightClearLogsRegexp.exec(logs)) {
      const switchId = parseInt(result[1], 10);
      clearRules.push(switchId);
    }
    return {addRules: newRules, clearRules: clearRules};
  }
 },
 MininetLogsParsers: {
  parse: async (switchesLogs) => {
    const map = new Map();
    switchesLogs.forEach(switchLog => {
      const result = switchLog.result;
      let parsedResult;
      const switchLogs = [];
      while (parsedResult = swithesRegexp.exec(result)) {
        switchLogs.push({
          src: parsedResult[1],
          dst: parsedResult[2],
          actions: parsedResult[3]
        });
      }
      map.set(switchLog.switchId, switchLogs);
    });
    return map;
  }
 }
};

const node_ssh = require('node-ssh')
const ssh = new node_ssh()
const readLastLines = require('read-last-lines');
const MININET_PORT = process.env.MININET_PORT || "192.168.56.103";
const DPCTL_PORT = 40100;
const SWITCH_AMOUNT = 8;
```

```
const PATH_TO_FLOODLIGHT_LOGS =
'/home/avorobiov/BACHELOR_WORK/logs/floodlight.logs'
const LINES_TO_READ = 3000;

module.exports = {
  FloodLightRetriever: async () => {
    return {
      retrieveLogs: async () => {
        return readLastLines.read(PATH_TO_FLOODLIGHT_LOGS,
LINES_TO_READ);
      }
    };
  },
  MininetRetriever: async () => {
    const sshClient = await ssh.connect({host: MININET_PORT, username:
'mininet', password: 'mininet'});
    return {
      retrieveLogs: async () => {
        let switchAmount = SWITCH_AMOUNT;
        const resultArray = [];
        do {
          const result = await sshClient.execCommand(`sudo ovs-ofctl dump-flows
tcp:127.0.0.1:${DPCTL_PORT + switchAmount}`);
          if (result.code === 0) {
            resultArray.push({switchId: switchAmount, result: result.stdout})
          }
        } while (--switchAmount > 0)
        return resultArray;
      }
    };
  }
};

const {FloodLightRetriever, MininetRetriever} = require('./retrievers');
const {FloodLightLogsParsers, MininetLogsParsers} = require('./parsers');
const EventEmitter = require('events');

const DEFAULT_INTERVAL = process.env.inverval || 10000;

module.exports = async () => {
  const floodLightRetriever = await FloodLightRetriever();
  const mininet = await MininetRetriever();
  return new Watcher(floodLightRetriever, mininet);
}
```

```
class Watcher extends EventEmitter {
  constructor(controllerRetriever, mininetRetriever) {
    super();
    this.controllerRetriever = controllerRetriever;
    this.mininetRetriever = mininetRetriever;
    this.cachedControllerRules = new Map();
  }

  watch = async () => {
    this._doWatch();
  }

  _mergeCachedControllerRules = (map) => {
    for (const key of map.keys()) {
      const receivedLogs = map.get(key);
      const logs = this.cachedControllerRules.get(key) || [];
      logs.push(...receivedLogs)
      this.cachedControllerRules.set(key, logs);
    }
  }

  _clearCachedControllerRules = (switchIds) => {
    for (const switchId of switchIds) {
      this.cachedControllerRules.set(switchIds, []);
    }
  }

  _doWatch = () => {
    setTimeout(async () => {
      const controllerLogs = await this.controllerRetriever.retrieveLogs();
      const mininetLogs = await this.mininetRetriever.retrieveLogs();

      const controllerResult = await FloodLightLogsParsers.parse(controllerLogs);
      const mininetResult = await MininetLogsParsers.parse(mininetLogs);

      const {addRules, clearRules} = controllerResult;
      this._clearCachedControllerRules(clearRules);
      this._mergeCachedControllerRules(addRules);

      const transactions = [];
      for (let switchId of mininetResult.keys()) {
        const mininetLogs = mininetResult.get(switchId);
        if (mininetLogs.length === 0) continue;
```

```
      const contollerLogs = this.cachedControllerRules.get(switchId) || [];
      mininetLogs.forEach((mininet) => {
        const index = contollerLogs.findIndex(({src, dst, actions}) => {
          return mininet.src === src && mininet.dst === dst && mininet.actions
=== actions
        });
        transactions.push({
          switchId,
          mininetLog: mininet,
          controllerLog: index === -1 ? {} : contollerLogs[index],
          result: index !== -1
        });
      })
    }
    this.emit("transactions", transactions);
    this._doWatch();
  }, DEFAULT_INTERVAL);
 }
}

const crateBlockChain = require('./blockchain/BlockChain');
const {createBlockChainClient, initBlockChainSyncer} =
require('./blockchain/BlockChainClient');
const {JsonMessageTransformer} = require("./blockchain/MessageTransformers");
const DigitalSignature = require('./blockchain/DigitalSignature');
const map = require('./conf');
const initWatcher = require('./flowrule/wathcer');
const nodeId = process.env.NODE_ID || "first";
const isFirst = process.env.NODE_ID === 'first';

(async () => {
 const messageTransformer = new JsonMessageTransformer();
 const nodeDetails = map[nodeId];
 const {id, host, port, nodes, isFirewall, privateKey} = nodeDetails;
 const digitalSignature = new DigitalSignature(nodes, privateKey);

 const blockChain = await (!isFirst ?
  syncBlockChain({messageTransformer, nodes, host, port, digitalSignature}) :
  buildBlockChain(digitalSignature)
 );
 if (isFirewall) {
  const initFirewall = require("./firewall");
  initFirewall(map, blockChain);
```

```
  }
  const watcher = await initWatcher();
  // noinspection ES6MissingAwait
  (await createBlockChainClient({nodeId: id, host, port, blockChain,
messageTransformer, watcher}))
    .withNodes(nodes)
    .listen();
})();


async function buildBlockChain(digitalSignature) {
  return crateBlockChain(digitalSignature)
    .generateGenesisBlock()
    .build();
}

async function syncBlockChain({messageTransformer, nodes, host, port,
digitalSignature}) {
  const syncer = await initBlockChainSyncer(messageTransformer, nodes, host,
port);
  return crateBlockChain(digitalSignature)
    .sync(syncer)
    .build();
}
```